A Dynamic and Adaptable System for Service Interaction in Mobile Grid

OTEBOLAKU ABAYOMI MORADEYO

(20067353)

A dissertation submitted in fulfilment of the requirements for the degree of

Master of Science

In

Computer Science

Faculty of Science and Agriculture
Department of Computer Science
University of Zululand

2007

DECLARATION

I, Otebolaku Abayomi Moradeyo, declare that this dissertation represents my work, and that it has not been submitted in any form for another degree or diploma at any University or other institution of tertiary education. Information derived from published or unpublished work of others has been acknowledged in the text and a list of references is given.

Otebolaku Abayomi Moradeyo

DEDICATION

This work is dedicated to the Almighty God for His love, Wisdom and Salvation that He freely gave me.

ACKNOWLEDGEMENTS

I would like to express my profound gratitude to those who made it possible to complete this research work.

First, I thank God Almighty for His Mercy and Grace bestowed on me in the course of my life and in particular during the period of this study. Thank you Lord for being the author and owner of my life.

Second, my sincere appreciation goes to my Supervisor, Professor M.O. Adigun, who has been the backbone of this work, and for the rare privilege he gave me to tap from his wealth of computing knowledge. In addition, you taught me what research is, and in particular, the art of writing. Your fatherly role can not be appreciated enough, thank you for believing in me.

I also acknowledge the financial support I received from the Centre of Excellence of the Department of Computer Science, University of Zululand without which this study would not have been possible.

I thank my family members- my Dad, my Mom and my Siblings for their support, I appreciate your care, love and patience.

My appreciation goes to my friends, colleagues and entire staff of the Department of Computer Science, Mr F. Nel, Mr Reuben Aremu, Johnson Iyilade, Dr Justice Emuoyibofarhe, Dr Xulu, TC Nyandeni, Stanley Ekabua, Petrus Shabangu, Mudali Pregansen, Miss Divya Pillai, Miss Tarirai Chani and others who contributed immensely not only to this work but to my life in general. The jokes, lively dispositions, love and the occasional frustrations we shared were inexhaustible fuel that propelled this research to a conclusion, thank you all.

Finally, I cannot forget my closest friend and love, Oni Comfort, who stood by me even when no one cared. I will eternally be grateful for the prayers, encouragements and love you shared with me in those trying times.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Mobile and pervasive computing with its peculiar feature of providing services at anywhere anytime basis has been at the centre of major computing researches in recent times. The device resource poverty and network instability have been reasons behind unsuccessful use of handheld technology for service request and delivery. However, interaction of these mobile service components can be adapted to further improve on the quality of service experienced by service consumers. Content, user interfaces and other adaptation mechanisms have been explored, but these have not provided needed service qualities.

However, one of the challenges of designing an adaptable system is on making adaptation decisions. This dissertation, therefore, presents a dynamic and adaptable system for service interaction. A context-aware utility-based adaptation model that uses service reconfiguration pattern to effect adaptation based on contexts was developed. It was assumed that developers of mobile services design services with variants that can be selected at runtime to fit the prevailing context situation of the environment. All variants differ in required context utilities. The service variants selection decision is based on a heuristic algorithm developed for this purpose.

A prototype of the model was built to validate the concept. Experiments were then conducted to evaluate the proposed model purposely to measure the interaction adaptation quality, the overall response time with or without adaptation, and the effect of service consumer preference for a given service variant on adaptation process. Results from the experiments showed that though the adaptation process comes with additional overheads in terms of variation in response time, the adaptation of service interaction is beneficial. It was observed that the overall response time increased initially as the number of service variants increased which was due to overheads by the adaptation process. However, as the number of variants increased, the response time began to fall sharply and then became steady. This proved that adaptation can actually help reduce service response time. We also found that adaptation quality degraded with increased number of service variants. The lesson learnt was that adaptation can help reduce overall response time and can improve service quality perceived by the service consumers.

CHAPTER ONE

INTRODUCTION

1.1 Overview

Mobile and pervasive computing with its feature of providing services anywhere has been at the centre of major computing researches in recent times. However, this work is particularly focused on context-aware dynamic adaptation of interaction between mobile Grid service consumers and Grid service providers, where the system monitors its own execution environment and reasons about such contextual changes that take place at any given time. The system then adapts on-the-fly to these changes in order to improve on the quality of services offered in the delivery of services to consumer in terms of service response times. The dissertation focuses on how this adaptation of interaction process can be controlled and effected at runtime. This chapter starts by motivating the need for dynamic adaptation. It then continues by giving a brief background on Mobile and Grid computing with challenges that call for adaptation. The chapter then establishes the problem this dissertation is addressing, including the goal and objectives set out to address this problem and the rationale of the research. The methodology for addressing this problem is then discussed. Finally, the organisation of the rest of the dissertation is then introduced.

1.2 Background

Computing is no longer limited to the desktop. Many different types of devices are increasingly taking advantage of the breakthroughs in wireless network technologies and the Internet to provide services to the global community. It is no longer news to see cellular or cell phones being used to browse the internet or used to access emails. Businesses around the

globe are setting up wireless networks so that their customers can have seamless access to information. All these are indications of a model of computing where increasing availability of small and smart devices with wireless networks provide consumers of services with convenience and functionality. The users of these devices want to be able to access services and carry out their computational tasks as they move from place to place regardless of the place, or time and the device they use, be it a PDA or mobile phone.

In this development, South Africa, which is not an exception to this breakthrough, falls in the middle tier of ICT development according to a report given in [78] with relatively large and growing rural communities. Furthermore, South Africans' access to information and communication services is very crucial to the livelihood strategies of her poor populace. The report says that mobile telephony in the rural communities is high with about 28% of the rural households owning a mobile phone compared with 29% of the urban communities. This is an interesting trend in the use of mobile devices in both rural and urban South Africa.

Therefore, the use of mobile devices among the disadvantaged population could be extended beyond ordinary telephony. A number of other services could be rendered to the rural communities through the use of mobile devices. These services have to be provided at low overhead cost and with good quality of service. These pervasive devices and their applications should be able to take advantage of facilities and information in the surrounding environment to provide relevant services. For example, a user might be working on a document using her PDA at an airport. She or he should be able to use a printer available in the airport directly, without having to go through setup procedures. Modern Mobile devices are equipped with features to easily discover other devices in their vicinity. This capability constitutes in a sense mobile grid computing infrastructure at an airport to enable the

discovery of nearby devices, services, etc., and provide links to them. In short, users want a seamless computing environment in which their device or their location is not a constraint as they perform their computational tasks.

However, the above need can be met if these services are crafted to cope and adapt in mobile environment considering the inherent limitations of mobile devices and wireless networks. Dynamic adaptation therefore, is a way to support these evolving execution environments [24]. It aims at allowing the applications to modify themselves depending on the available resources and changes in their context of execution. The key idea here is that as the context of application changes, it should be able to modify its own function or interaction in response to these contextual variations.

Software systems of today are, however, continuously growing in size and complexity and are not designed with adaptation in mind. Very recently, these applications are migrating from their traditional (desktop, fixed network) executing environment to highly mobile, distributed, pervasive and Grid computing environments. These environments come with inherent challenges that tend to defy solutions [1]. These result from the features of the devices and networks in which these applications are designed to execute.

Notable among the characteristic features and challenges of mobile systems are: limited Central Processing Unit (CPU) power, limited memory, small screen, short battery life, heterogeneity, low bandwidth of wireless network and intermittent disconnections [2].

One of the suggested solutions is the use of mobile devices to access, provision, share and perform on-demand service delivery from the Grid [3]. Today, far beyond the capabilities of other existing technologies such as Internet, the Grid enables the provisioning, accessing,

sharing, adding, removing and managing of resources and services such as storage, processing, network, and database and software application resources. These resources are dynamic as the demand for them is not static. The Grid has become first choice for problem solution in science, business and commerce [3]. It adopts different features from existing paradigms such as clusters computing, utility computing, autonomous computing and peer-to-peer computing.

Most applications today are not designed to run on the Grid but they only need slight modifications to be Grid ready. A recent development has caused some researchers to develop a keen interest in the implementation of Grid concepts [4] in mobile environments. This is particularly articulated in the use of mobile devices in submitting jobs to the Grid, retrieving jobs and delivery of services. Adaptable Grid applications that can adjust to both the dynamism in the Grid environments and the wireless network that mobile devices will use to access the services offered by the Grid should be the goal to achieve full-fledged mobile Grid adoption. These, according to Kola et al [12], are needed for two major reasons. One, applications could be multiphase, dynamic, and heterogeneous requiring large number of software components with complex interactions. Two, the mobile Grid infrastructure is also dynamic, heterogeneous and changing due to the inherent features of mobile devices and ubiquitous network that they use to access the Grid. These two factors do put major burdens on service composition, accessibility, sharing and delivery. Adaptability, therefore, needs to be enforced at multiple levels from the application running on the mobile devices to the network, computation, and storage resources.

These services must be composed in such a way that they can adapt themselves to changes such as resource unavailability, network latency, bandwidth variations, interaction

dependencies and link failures. The interaction latency adaptation which is what this research set out to investigate is inevitable in services that cannot tolerate long response latency. Primarily, this work investigates how to achieve reduction in response time experienced from service consumers' interaction with service providers. Thus, the problem to be investigated assumed that:

(1) There are many service consumers each making use of some computing devices such as mobile phones, Personal Digital Assistants (PDAs), and Personal Computers (PCs) connected to a distributed computing infrastructure such as the Grid,

(2) Each service may likely be distributed over the computing infrastructure,

(3) The service consumer's devices have very limited resources such as memory, processing power, which are not enough to execute the requested service,

(4) Devices are connected to the distributed Grid computing infrastructure by an unstable wireless network with varying bandwidth,

(5) The service consumer service preferences, device resources and environment change dynamically.

Thus, methods and approaches need to be explored in this regard to make mobile systems cope and adapt to disruptions in their execution environment that will result in reducing ultimately the dissatisfaction experienced by service consumers owing to these limitations. This could be achieved at two levels namely [8]: at the underlying system level and at application level. In the application transparent approach, the underlying system is solely responsible for adaptation. This is beneficial as it causes the application to run uninterrupted. It is, however, not desirable if there is a long period of disruptions. In application aware adaptation, the application collaborates with the underlying system to adapt. The underlying

system provides the application with some status information which it uses to make some adaptation decisions according to changes in the resource availability and mobility. The systems have to adapt to the changes in the environment such as the network configuration and availability of the computational resources and services. This adaptation could be performed by employing some mechanisms. One, adapting the data being accessed by various components of the system by varying its quality and two by adapting the functionalities of the system components.

The first mechanism involves considerable overhead as it compromises the system performance. The second mechanism however, according to Kristler et al [8], involves changing dynamically the functionalities of the computational entities in response to changes in operating conditions. These functionalities could be adapted according to Mikic-Rakic and Medvidovic [21] by:

(a) Making remote data available locally,

(b) Making remote code available locally,

(c) Making a remote dynamic system state available locally, re-routing the communication in cases of partial disconnection from the network, and

(d) Delaying remote interactions until the connection is re-established.

The purpose of the above is to temporarily mask the users from the absence of connection by mimicking the system's continuous connection. These could be achieved traditionally by employing some adaptation techniques namely:

(i) Caching – locally storing remote data once it has been accessed in anticipation that it will be needed again,

(ii) Hoarding – pre-fetching the likely needed remote data prior to disconnection,

(iii) Queuing remote procedure calls – buffering remote, non-blocking requests and responses during disconnection and exchanging them upon reconnection,

(iv) Deployment and redeployment – installing, updating, or relocating a distributed software system,

(v) Replica reconciliation – synchronising the changes made during disconnection to different local copies of the same component,

(vi) Code mobility – dynamic change of the bindings between code fragments and locations where they are executing.

Some other mechanism is the adaptation of the systems functionalities at architectural level using software architectural principles [21]. Software architectures provide high level abstractions for representing the structures, behaviour and key properties of a software system. But this comes with a lot of complexity which is not cost effective.

All these techniques, however, came with a lot of sacrifices in terms of correctness, scalability, message throughput and bandwidth consumption that are not desirable. Therefore there is a need to consider some more efficient strategies.

1.3 Statement of the Problem

Some careful and critical investigations into the various approaches currently in use to address major challenges in the design and development of adaptable and reconfigurable mobile Grid applications have identified the use of software architectural principles and component based software engineering to address the problem of disconnection operation and dynamic reconfiguration resulting from contextual variations in a mobile environment.

Zuma and Adigun [5] proposed an adaptive interface pattern for mobile applications in which they defined the interaction pattern between context data producing components and context data consuming components in a mobile distributed application. Zuma and Adigun proposed a protocol for the serving of context data among interacting components of a mobile application that are possibly distributed on various hosts in an unstable network. They did not address adaptation of these components to context variations in the execution environment. Zuma and Adigun in [5] further described an interaction architecture that defines a protocol for providing context information and other elements of a mobile application. Shared space architectural style and component based development were used to design this architecture. This goes a long way in addressing the disconnected operation and the quality of service adaptation at the higher abstraction level.

However, the adaptation and reconfiguration of mobile clients in order to specifically reduce the consumer's dissatisfaction due to resource starvation, mobility and dynamic nature of the underlying wireless network is not yet properly addressed. So, how do we design and develop a mechanism and a strategy for adaptable interaction between mobile clients and Grid service endpoints in order to reduce the service request response?

1.4 Rationale of the study

This work is a contribution to the Context Aware Component Interface Pattern (CACIP) and is influenced by various other research works being conducted in the field of mobile and Grid computing [4] and component based software engineering [22]. These researches, recently, are being carried out to address challenges facing application development for mobile environments. Zuma and Adigun in [5] proposed a Context Aware Component Interface Pattern that defined the interaction pattern between context data producing components and

context data consuming components in a mobile but distributed application. The CACIP provides a protocol for serving context data among interacting components of a mobile application that are possibly distributed on various hosts in an unstable network bandwidth. This interaction pattern has been adopted (by the centre) for the newly crafted Grid utility based infrastructure service technology (GUISET), which is a software service driven e-Service technology based on emerging concept of utility grid computing. In this infrastructure, the interaction between the mobile client components and the server components posed serious challenges partly because of instability of the wireless network connecting these components and partly because of the heterogeneity of the mobile devices. However, CACIP specifically did not address how mobile service components in this infrastructure could cope under patchy connections in the network, in which they run and assumed a shared address space, which serves as a performance bottleneck with regard to scalability. This research work is therefore, motivated by the quest to propose a model that can help components of mobile application services adapt to its unstable environment in order to reduce the overall response time between a service consumer and service provider.

1.5 Research Goal and Objectives

The goal of this research was to propose an adaptable and reconfigurable interaction and communication system between Grid clients and Grid services.

In order to achieve this, the following issues were addressed objectives to:

(a) Develop suitable adaptation mechanisms and algorithms that will help to reduce service response time between mobile Grid clients request and service delivery,

(b) Craft a context-aware utility-based adaptation conceptual model.

(c) Design a decision making algorithm based on the developed conceptual model,

(d) Develop a prototype of the crafted conceptual model that was evaluated and validated through a number of experiments.

1.6 Research Methodology

The adopted methodology focused on challenges of successful interaction between mobile clients and Grid services. The following approaches were followed in our quest to solving this adaptation problem.

1.6.1 Requirements Gathering

Investigations on various adaptation mechanisms for mobile systems were carried out. This was necessary for the understanding of the state of the art adaptation mechanisms already in use. These investigations were specifically focused on the following research areas namely context awareness in mobile systems, adaptability and systems reconfiguration. The knowledge garnered here was adapted to arrive at methods for developing and evaluating appropriate adaptation mechanism to guide the design improvements for the CACIP.

1.6.2 Model Development

First, the CACIP model was fitted into the mobile Grid scenario. Second, an adaptation model for the CACIP was developed in order to reduce the delay experienced between service request and service delivery. To achieve these, the model was able to sense service executing environmental and device contexts. It was also able to reason about these contexts dynamically in order to take intelligent decisions on the prevailing contexts at any given time. Finally, the model was able to reconfigure the service dynamically.

1.6.3 Simulation and Evaluation

An evaluation of the proposed model was carried out. In doing this evaluation, simulation of the model was first carried out. This was conducted with and without the adaptation mechanism. This was done to see how the model achieved the set objectives. A number of other experiments were conducted to ascertain the effects of adaptation, service consumer choice and number of service requests.

1.6.4 Organisation of the Dissertation

The remainder of this dissertation is organised as follows. Chapter two presents an in-depth state-of-the-art review and theory of dynamic and reconfigurable service adaptation. It also presents review of very recent works in software adaptation, classifying them based on the adaptation strategies, adaptation mechanisms, decision mechanisms, context-awareness and their adaptation goals. This helps to identify gaps which our work attempted to fill and to identify which of these systems will be of help in our design and implementations. Chapter three describes the design of the dynamic adaptation model. The chapter begins by discussing the aims and objectives of the project followed by the illustration of an example scenario. From this discussion some requirements for the dynamic adaptation model are established. This chapter then continues with an in depth discussion of the context aware and reconfigurable model, as some of the concepts adopted in the design were explained. The decision making model based on a utility function is presented with an algorithm that helped address this problem.

The chapter ends with a discussion of how the dynamic adaptation model design presented in this chapter fulfilled the objectives and requirements in order to support the dynamic interaction adaptation of mobile grid systems in a context-aware manner. Chapter four

presents the implementation of a prototype based on the resource management and dynamic component model of the MADAM adaptation middleware [25]. The chapter then continues with the design and simulation of a hypothetical multimedia service request which was used to evaluate the performance of the model. The result and its implication on the model were then presented. Chapter five concludes the dissertation with a discussion on how the stated goal and objectives of the project were achieved. Also, limitations of the model based on the evaluation performed are then presented which are used to give possible future works to improve on the proposed model.

CHAPTER TWO

BACKGROUND CONCEPTS AND REVIEW OF LITERATURE

2.1 Introduction

This chapter provides some background on the research areas with which this dissertation is concerned. Sections in this chapter analysed the most influential researches in the area of dynamic context-aware adaptation and adaptation management. Key areas of interest include dynamic adaptation, context-awareness, service reconfiguration, and policy-based management of adaptable software. Since the area of adaptable software is very wide ranging, the set of systems described is the set of most influential systems, not completely an exhaustive list. They were however analysed specifically with respect to their relevance to the aims and objectives of this dissertation.

The chapter is divided into four sections. The first section provides an overview of mobile and pervasive computing, while section two discussed Grid and Mobile Grid computing; section three gives some insight on adaptation and context awareness. Finally section four gives an analysis of some adaptive systems in relation with their adaptation approaches, strategies or mechanisms which helped to identify gaps in adaptation for mobile and pervasive Grid that satisfy the goal and objectives of this research. The chapter concludes by an overview of the analysed systems and some open research questions that this work attempts to address.

2.2 Mobile and Pervasive Computing

As mobile technologies are becoming integral parts of our society and working environment, the increasing pervasiveness and mobility of computing and communication enables new services and applications that can improve the quality of work and life. Two trends are highly-significant in modern computing technology with very rapid development in the areas of pervasive and mobile systems. One, millions of mobile devices are being deployed, and two, more integration and computation power are required behind the scenes to provide opportunities in new application domains. However, accessing services provided by some end systems using mobile devices is a challenging objective because such small devices are typically resource-constrained, with limited processing, memory, storage, energy and network resources [38]. Among the solutions proposed for this problem is to design an infrastructure that adapts these services and mobile devices to the prevailing contexts of their execution.

Foreman and Zahorjan of the University of Washington [54] defined mobile computing as a technology that enables access to digital resources at any time, from any location. The traditional desktop computing and wired network were very static; they restricted their users both in time and space. However, the idea of mobile computing frees the users from these restrictions. A mobile device user can use her system without space or location restriction, and enjoys the satisfaction she has when she uses her traditional desktop system. Before now, if a user wanted to access her electronic mails, she would do this in a confined environment. If she was not in her office or at home where she probably had desktops, she might not have access to the mails except if she got to where a desktop that was tethered to the Internet was located. However, with the convenience and ease mobile computing has brought, coupled

with the advancement in the wireless network technology, the proliferation of intelligent portable computing devices (e.g. cellphones, PDA and laptops), a user can access information on-the-fly anywhere, anytime [58]. This use of portable computers with wireless network has revolutionised the way computers are used. Pervasive computing extends this capability of migrating from fixed desktop computing to mobile computing platforms by enabling users or consumers of services to have access to these services in a way that customises the services to specific service consumers' request and the task at hand[55].

This vision has led to two fundamental features of pervasive computing: The mobility and context-awareness issues that result from the extremely dynamic nature of mobile and pervasive computing environments. Mobility according to Nalini [38] comes with two dimensions. First, the mobile application needs to run on varieties of devices including those embedded in various environments and those ones carried by users. Second, these devices are mobile and are connected by an unstable low bandwidth wireless network to a more stable fixed network. This means that applications designed for such an environment must be made to run in degraded mode as a result of varying and unstable network bandwidth. Sometimes, this network is not even available, yet a pervasive system user expects an all-time availability. This application must also be very sensitive to its environment as the context changes. Context-awareness [47] is important for pervasive computing since environmental contexts change dynamically, the system must be aware of these changes so that it can make some appropriate and feasible decision on its context of execution. For a pervasive system to be context-aware there is a need for it to identify and bind to sensors that provide data whenever changes occur in the environment [25]. A means to compose information from these data in order to create useful information for decision making will also be needed.

Furthermore, pervasive systems can be aware of three types of contexts: The mobile device contexts such as memory, CPU processing power, and storage; the network contexts such as bandwidth, network types, network configurations and user contexts such as location, user preferences profiles and QoS specifications.

However, mobile and pervasive computing paradigm came with serious technical challenges that are yet to be resolved. Device heterogeneity is one of the problems faced by the technique. End user devices- smart phones, PDAs and embedded sensor computers come in varieties with varying capabilities both in hardware and software. The form factor, the processor, the memory and networks vary for each device vendor. The operating systems, the services and application developed for these devices are of course diverse [33]. Hence, the impact of device heterogeneity is that application and services need to be developed for each device. This means that services and applications developed for a particular device may not execute on a different device. Wireless network issues are also a serious source of nightmares. The network bandwidth is not always enough for transmission of services from service endpoint to service consumer's devices. But effort is already being made to make pervasive services to adapt to these various problems so that such service users or consumers can have these services with satisfactory quality of service.

2.3 Mobile and Pervasive Grid

Section 2.2 gives an overview of mobile and pervasive computing which recently gave birth to mobile and pervasive grid. The growth of the Internet with the availability of powerful yet small computers and high-end fixed networks along with great reduction in the cost of these resources have gradually changed the way computing is done. Various technologies [58] have enabled clustering of a wide variety of geographically distributed services and resources.

These resources among others include supercomputers, storage systems, data sources, special devices and services which are aggregated for purpose of having virtual, powerful computational resources. These resources and services are provided with seamless access and interaction among the constituent components. This new approach to computing is termed Grid.

One of the recent visions of this distributed computing paradigm is to readily make services, data, and resources available to their consumers the same way electrical power and other utilities [9] are made available to us. This vision has evolved into provisioning of service oriented infrastructures that leverage standardised protocols and services for the reason of pervasive access, and coordinated sharing of geographically dispersed resources and services. These potentials for seamless access, aggregation, integration and interaction created keen interests among science and engineering communities to conceive a new paradigm of distributed computing that helps investigate complex scientific and engineering problems. Grid computing then can be summarily described as computing infrastructure where computational power and resources such as services, data, CPU speed, memory, etc. is as readily available as electrical power. These computational services make this power and resources available to consumers with differing levels of expertise in diverse areas and in which these services interact to perform specified tasks efficiently and securely with minimum of human intervention. While accessing these services is on on-demand basis and ubiquitous, they can as well be dynamically and transparently constructed from distributed sources. Therefore, the service consumers do not need to know prior to accessing services through a simple mobile terminal. Service providers on the other hand can extend Grid services facilities at any moment, as it also manages the architecture and defines policies and

rules for accessing and consuming such services. Hence, according to Foster et al [9], the Grid computing is an emerging way of thinking of a distributed environment as a global scale infrastructure to share data, distribute computations, coordinate works, and access remotely and geographically dispersed services and resources.

However, this original vision of the Grid has moved from what it was in a new dimension. The explosive growth in computation and communication infrastructure together with proliferation in the mobile device technology has given birth to a new Grid concept known as mobile or pervasive Grid. This is the marriage of mobile or pervasive computing with Grid computing. Mobile computing provides the ubiquity and mobility for accessing conventional Grid services while the Grid provides the bridge for mobile devices in terms of storage, computation, resource discovery, and data scheduling and processing power. However, wireless network and mobile device limitations have to be integrated seamlessly into more stable and resource-rich Grid infrastructures. The need for this, therefore, calls for a way in which mobile services running on mobile devices that are designed to access available services in the infrastructure can be made to adjust and fit into such dynamism in its execution contexts. In the section that follows, a description of the management of such technique known otherwise as adaptation is presented.

2.4 Dynamic Adaptation Management

From the preceding sections, it can be seen that there is a fundamental need to integrate both mobile systems with Grid infrastructures in a manner that will seamlessly make services available to service consumers with good quality of service. Service adaptation can be described as changing a service according to context changes, including user needs, which influence its execution in order to make the service available in the face of the adverse

context changes.

Though these contexts change, some of their properties particularly their core functionalities remain the same. It is currently the focus of extensive research [48]. Some of the techniques such as data adaptation, energy-aware adaptation and context-aware adaptation have recently become crucial in achieving adaptation [38]. We want services to be able to change at run-time, say, to adapt the amount of memory available for their use, the network bandwidth they take up, or maybe, when running out of power, switching to low-power mode. They should be able to detect these changes in the environment and respond to them appropriately. This is the end result we want services to achieve. On the other hand, application services can employ various ways to respond to the change – "the means". By changing quality of the data accessed, such as a poor quality image, both network bandwidth and memory can be saved. In other words, we are adapting the data quality, in order to achieve two ends.

It follows that on-line adaptable systems must react to fluctuating environments. For instance, if users get connected to wireless multimedia telecommunication services during peak period, dynamic adaptation maybe desirable rather than dropping calls or rejecting packets arbitrarily with no care about the rendering. The idea here is to adapt the functions or behaviours of the systems to the fluctuations in their environments without apparent degradation. This will involve dealing with adaptation reasoning and determining what system changes to perform. In the next section, existing adaptation mechanisms and policies are classified.

2.4.1 Adaptation Mechanisms

In order to design and model adaptation, researchers have used various adaptation mechanisms and policies. Adaptation mechanism deals with a way to support potential adaptation and the adaptation policies deal with semantic-based adaptation tailored towards the underlying applications [25]. Some of these mechanisms are briefly discussed in the next section.

2.4.1.1 Migration or Code Mobility

The cited works in [22, 27] deal with movement of distributed application components at runtime. One can also say that it involves moving service from one service node to another in order to reduce transmission distance and overheads between a service consumer and a service provider. However, migrating service components from one node to another, requires the need to preserve states of the migrating service.

2.4.1.2 Parameter adaptability

This involves modifying variables that determine program behaviour. In mobile computing, this approach has been adopted in context-aware systems for developing pervasive environment [25]. Here, applications are tuned using some external environment properties, for example this has been used to adjust a TCP protocol by controlling parameters that influence retransmission in response to apparent network congestion. However, parameter adaptation is not suitable for handling configuration that is not decided at design time, but offers the advantage that adaptation help to achieve good performance.

2.4.1.3 Compositional Adaptation

This is a kind of adaptation that results in parts of a system or algorithm being exchanged in

order to improve the application's fitness to its environment [30]. The larger adaptation scope of this mechanism gives it an edge over all other adaptation mechanisms. It does not only enable simple tuning of code programmed for design time but also copes with such adaptability that is not anticipated during system's original design and development. This type of adaptation redeploys and recomposes applications or services while taking into consideration the context of execution of such services.

In [30] compositional adaptation mechanism was defined by answering these three questions.

(a) How to compose adaptation: In order to answer this question, some specific approaches used to compose adaptation are discussed next. Techniques such as reflection, software components composition [30] create some level of indirection between application entities to construct open and reconfigurable systems while applying the separation of concerns principle, which means separation of functional and non-functional aspects of the applications.

A software component based approach is a good paradigm used for composing adaptation. Services are constructed by composing a set of components both statically at design time and dynamically at run time. At design time, components of a service are specified like contracts as component types and interaction between these components as connectors.

The service components model is very useful in handling adaptability of mobile systems. Many component based systems have adopted this pattern. Systems such as SATIN[29] define component models that support logical migration, dynamic reconfiguration using active rules; while some use planning as an adaptive deployment process to select appropriate component composition.

(b) Where to compose Adaptation: Compositional adaptation can be constructed in two different ways; dynamic and static compositions. In dynamic composition, there are mechanisms that can be applied at run time such as removal or addition of components at run time without halting or restarting the application. It is a potential way to compose dynamic adaptation. However, static adaptation takes place at design time i.e. at development, compile and load time. In this approach, adaptation is hard-coded into the application codes. This means that the adaptation behaviour cannot be changed at run time. Some techniques such as Aspect Oriented Programming using Aspect J [46] have been used in this regard.

Reflection is the ability of a program to reason about, possibly alter its own behaviour [66]. Reflection distinguishes between the base level that represents run time system objects and the metal level that reflects the base level transparently. These two levels are causally connected, meaning that modifications to either of the two will be systematically reflected on the other level in both directions. Two types were identified: ***Introspection and Intercession*** [25]. In introspective reflection, architectural properties of the running application are observed and sensed including interaction between program entities, method signatures and states, but intercession enables a system or an application to act on these observations and change its behaviour. This reflective technique has been adopted in a number of mobile middleware applications to observe and reconfigure the system according to the adaptation needs and context changes [66].

(c) When do we compose Adaptation? An answer to this question deals with the level of the system at which adaptation can be integrated. This maybe in the code of the application itself or a separate system maybe used to achieve adaptation such as in the middleware approach [30]. Three different levels have been identified [33]. At one extreme, adaptation is

independent of the systems (both operating systems and middleware). This approach is also called the Laissez faire approach [33]. This approach embeds the adaptation into the application code and hence makes the application development so difficult and complex. It lacks resource arbitrator that helps to resolve resource demands by the application and the available resource. The application transparent is another extreme approach where the systems (both operating systems and middleware) are solely responsible for this adaptation. In this technique, adaptation requirements of the application are not taken into consideration. Between these two extremes lies the application aware approach where adaptation is a collaborative responsibility of both application and the systems. The systems supplies information on the underline context and resources while the contexts of the users and its devices are also provided for appropriate and dynamic adaptation decision.

2.4.2 Adaptation Policy

Designing mechanisms or strategies to handle adaptation, or to reconfigure a mobile system is not enough. The most complex problem in the field of dynamic adaptation is the problem of how to calculate or derive a new feasible configuration that will fit the current situation in systems executing contexts. The main advantage for using policy is the separation of concerns between functional and non-functional properties of an adaptive system. The declarative nature of policies simplifies the definition and change of adaptation strategies. Depending on the domain in which adaptation is being designed, such adaptation policies can be realised in different ways. In [25], three kinds of policies were distinguished. The action-based, the goal-based and utility-based policies.

2.4.2.1 Action based polices

Action based policies are very popular and are used in many domains such as network and

distributed systems. These types of policies consist of situation-action rules that specify exactly what to do in certain situations. This approach was adopted in the domain of software architectures to express dynamics of system architectures. These rules sometimes are expressed at the architectural description language level (ADL) [71] by associating invariant in the form of event-action rules so as to express dynamic reconfiguration actions in components-based architecture. A good example of systems that explicitly used the concept of an action based adaptation policy is DART [72]. Its policies were represented and associated to components, coordinated and managed by a manager component. The coordination included the resolution of conflicts and incoherencies between the set of policies present in the system. In order to handle these, the DART's policies were organised at three abstraction layers: the system, the middleware and the application. At each abstraction or level, the policies are further organised with different priorities. Safran [67] and Chisel [27] extended this idea by proposing self-adaptive component model and adaptation manager as separate entities. This enforced separation of concern from application functionalities. Safran[67] and Chisel[27] for instance, extended the Fractal component model by associating rule based policies to fractal components as a new kind of components controllers for adaptation.

Though these systems allow integration and modifications of policies dynamically, they did not make provision for the policy management problem in situations where there are many applications with different policies in general and distribution in particular.

The action based policy though very powerful; its use in pervasive environment is a very difficult venture. Apart from the fact that such policy managers must be very familiar with

low level details of system functions; its use becomes very complex. It may also prevent some systems from exhaustively exploring all adaptation options.

2.4.2.2 Goal-based Policies

This is an AI based approach to adaptation. Multi-agent systems and planning algorithms are good examples of areas where this approach has been embraced [25]. In goal-based policies, a higher level form of behavioural specifications that established performance objectives that left the middleware to determine the actions required to achieve adaptation objectives were designed. This is particularly useful in systems that determine algorithms that will allocate computational resources to guarantee some QoS. Goals provide binary classification of policies into either desirable or undesirable performance, which is to either maximise the probability of achieving such desirable state or minimising the undesirable state. A major setback of this policy approach is that it does not offer a mechanism and flexibility to measure how one solution is appropriate to one situation in order to negotiate contracts between competing mobile adaptive systems. A good example of systems that adopted this type of policy is the work by Doyle et al [70].

2.4.2.3 Utility Function Based Adaptation

This is an objective function that expresses the values for each current state of the systems. This function permits on-the-fly determination of a best feasible state while other policy types place the system at any state that are both feasible and acceptable at that point without any provision to improve the system overall performance[25]. Some existing systems [25, 28, 29, 32, 37] use utility functions to qualify and quantify the desirability of different adaptation alternatives. These works are quality of service based and actually deal with resource allocation and typically in mobile and pervasive computing. Systems such as QuA

[29] and Odyssey [8] are examples of systems that utilise utility function based adaptation policies.

2.4.3 Adaptation Strategies

An adaptation strategy deals with what needs to be modified or adjusted and how to go about this [38]. Adaptable applications can change their runtime characteristics in response to some external triggers or changes. These runtime changes are considered using several dimensions to categorise adaptation strategies as discussed in the next subsections.

2.4.3.1 Network Adaptation

This is the ability of systems to change their network behaviours in response to changes in the network infrastructure. They should be able to reduce their bandwidth requirements, accept a greater degree of packet loss or be able to connect to a new network environment as the need arises.

2.4.3.2 Memory Adaptation

This is the ability to adapt to the changes in run-time memory available. Every device has different amounts of memory and processing power available. An application should have the ability to provide the same functionality in a resource-rich PC and in a resource-constrained PDA environment. Moreover, with several applications running at the same time, it is possible that one application may find itself suddenly out of memory. In that situation, it should gracefully adapt to perhaps a more memory efficient mode.

2.4.3.3 Energy Adaptation

This is the ability of an application, or system, to adapt its energy usage. Power is one of the most limited resources in a mobile environment. Applications and systems should be able to run in a more energy efficient manner, in situations of limited power supply.

2.4.3.4 Device Adaptation

This is the ability to adapt to device configurations. Users move from one device to another. Applications will need to follow the users. The devices may have different input and output capabilities. For example, a PDA may use a pen-based input, whereas a laptop uses a keyboard. They may have different processing powers. It involves more than just changing the device drivers. The presentation format, the UI and perhaps the application logic also need to adapt to the new configuration. Applications need to provide a seamless transition from one device to another. A good example of system that provides device adaptation is MADAM [30] where an application can be adapted to the screen, speaker, and network adapters of the device.

2.4.3.5 Data Adaptation

Mobile applications usually need to access data for information or for entertainment, such as emails, stock quotes, web pages, multi-media, etc. Data adaptation involves changing the data in some manner, such as changing the quality of the data accessed, transforming data to a more appropriate form, accessing a different set of data altogether, etc. This is the basis of many of the transcoding and content adaptation techniques. There are several projects that use proxy-based data adaptation to change the quality, or fidelity of the data accessed, on the fly according to the client resource available, or according to the network environment [67]. For example, in Odyssey [8], the server has several pre-generated versions of the data with different fidelity levels, and the appropriate one is chosen at run-time according to the resources, or even energy, available.

2.4.3.6 Functionality Adaptation

This involves changing the way an application carries out its functionality. Applications in a

mobile environment can be seen as fulfilling certain tasks [5]. Functionality adaptation implies carrying out the same task but in a different manner, either by using a different mechanism, different algorithm, a different QoS characteristic, or by switching to another execution mode, etc. It involves changing the execution of the task. For example, if a device does not have sufficient computation power, an application can use a smaller key for encryption.

2.4.3.7 Migration Adaptation

This involves changing the location of execution, for example, by moving to another machine with more resources. This is often used in the fields of distributed computing and mobile agents, whereas in the field of mobile computing, it is still rare or underdeveloped. Adaptive distributed applications and mobile agents migrate to nodes, which fulfil the resource requirements, if they realise that the current node does not have sufficient resources for their execution [18, 33]. Migration adaptation can be advantageous in a mobile environment, for example, migrating from a PDA to a nearby laptop in order to speed up execution.

2.4.3.8 Interaction Adaptation

Being the focus of this investigation, interaction adaptation deals with adjusting the communication between ports and connectors of a given mobile application either as a local or distributed application. Such systems are component based and have their functional and non-functional properties described in their architecture. A good example of system that adopts this adaptation mechanism is Prisms [21]. The interaction adaptation may involve disconnecting or removing, or adding of components, connectors and ports in order to adapt the system to its current executing contexts.

2.5 Context Management

The previous section discussed adaptation management with various mechanisms and strategies in existence that has been explored in adaptive systems. In this section context management is discussed. One of the primary requirements for achieving or designing context aware adaptation model is that the model must make provision for sensing the changes that occur in its context of execution at runtime. Context management maybe referred to as the collection and management of contextual information that enable context aware computing. The context aware computing paradigm does not only adapt applications but also adapt to changes in context that occur during the execution of the application such as time of day, location of the system. Dey [26] defined context as "any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the application and the users.

In this work, context is defined as those situations surrounding executing systems that either positively or negatively influence the system. An adaptable mobile Grid system is expected to provide appropriate mechanisms for acquiring, storing, retrieving and evaluating contexts. The evaluation of contextual information is a triggering process that causes the execution of adaptations. The context management aims at accumulating, processing and forwarding of relevant context information. The following section is focused on the major concerns in implementing context aware systems- one of the characteristics that must be built into the adaptation model in this research. To understand this concept, the following questions are addressed together with analysis of context aware systems. What context data should the

systems sense and present? How does the system collect these context data? And how the context data should be modelled and stored? Context aware systems have the capability to discover and exploit contextual information such as user and system locations, time of day, nearby people and devices; user activities and environmental contextual information such as network QoS parameters. Contextual information of context aware systems infrastructure is the core issue that must be addressed to be able to implements such systems.

It is obvious that the mobile and pervasive environment is unpredictable whilst the application is moving between different geographical locations, different supporting devices, and variable and local parameters such as noise, power level, user preferences etc. Therefore there are needs to accommodate various types of context data structures and means to handle them appropriately. In [25], context information was categorised into:

(a) Physical Context

Noise, light, temperature and traffic conditions are some good examples of physical context. These are context information that describes environmental factors that can be sensed by using some specialised hardware devices such as light and noise sensors.

(b) Computing Contexts

This is related to computable or measurable information that is retrieved by calculating or checking some routines. Information such as the memory of mobile devices, its processing power, and storage. Also, network connectivity, bandwidth, latency, and throughput and nearby resources are good example of computing contexts. This context information can be sourced by using either logical or virtual sensors that source context data from databases logins etc.

(c) User Context

User profiles, preferences, interests, expertise, workload, tasks, etc, present good examples of user contexts. This is closely related to the user personalisation issues that enable the ability to adapt products and services either to large user groups or, smaller interest group or individual users.

2.5.1 Context Sensing

Context information can be retrieved by using some specialised mechanisms which are often referred to as sensors [53]. Time for instance is sensed by using a system local clock to check scheduled tasks or current activities. Systems such as Active Badge and Cyberguide [64] use time to adapt the systems behaviour. Location is sensed by using a Global Positioning System (GPS) for outdoor location sensing. For indoor, transmitters and receivers are used. This location information can be used to adjust systems behaviours and a good example is a call forwarding application that uses location information to find the nearest phone to the users. Network bandwidth can be sensed using kernel functions that measure bandwidth and gives notification to the adaptation subsystems.

2.4.4.2 Context Modelling

In order to handle contextual information that is produced by the context sensors, it must be stored in data structures that make it easier for retrieval. Different ways have been explored

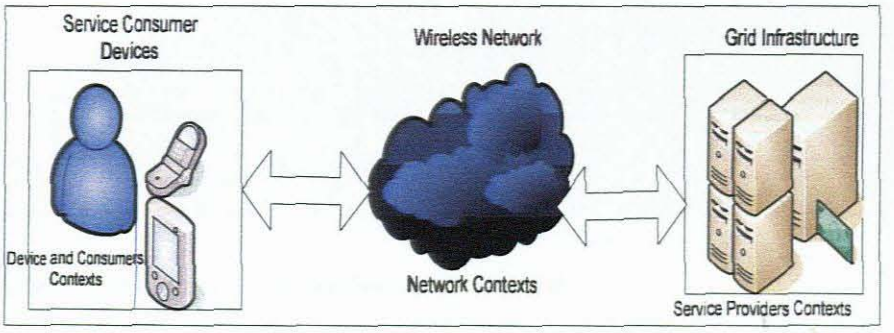

Figure 2.1: Illustrating Mobile Grid Service Execution Contexts

to model and express context data. A service provider in Figure 2.1 is considered as an entity in the Grid system that provides some communication services, storage services, processing services, or any service as utility to service consumers. It is envisaged that these services are provided as utilities to service consumers on demand [56]. The access to this contextual information, therefore, provides an opportunity to use context in service provisioning that would result ultimately in some higher quality of service.

Most of the existing traditional Grid systems perform their functionality based on explicit input and are not necessarily aware of the context of their execution. For example, a web service request would be made based on the service URI.

The result will be the service being loaded into the client's device. This result may not well be well presented on the client device if the service provider is not provided with contextual information that tells it whether the client device is a mobile phone with constrained resources or a desktop with considerable rich resources. However, in a context aware environment, this information is integrated as part of the service or an external context-aware module monitors the context of the system. Figure 2.2 illustrates a service environment that

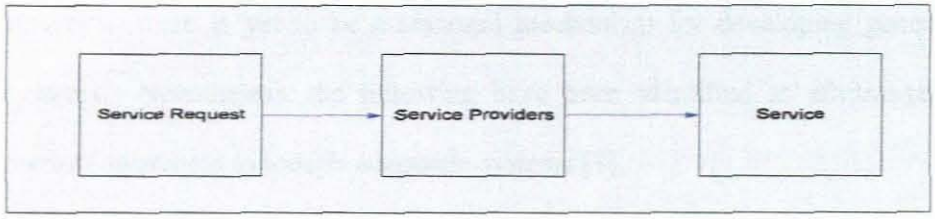

Figure 2. 1:Context-Unaware Service Provision

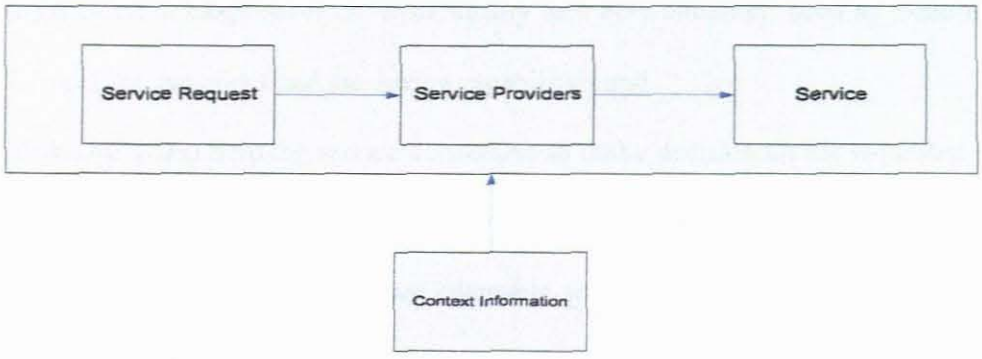

Figure 2.2: Context-Aware Service Provision

is not context aware while Figure 2.3 illustrates a context-aware service environment. The context-awareness is built as an external module. For this type of systems to provide context-aware services there is need for context input besides functional input.

This means that the output of such a service request depends on the context information the system is provided with. This contextual information is provided by some intelligent sensors and monitors. These contexts can be categorised into three: One, the service client's context such as available memory, storage and processing power of the mobile clients; Two, the environmental context, such as network bandwidth, latency, network availability, etc and finally, the service consumer's contexts.

These may include, among others, user profiles, services request types and some QoS specifications. These are explained in more detail in the next section. In a web service for instance, some information about the service can be transmitted using the SOAP header.

However, there is yet to be a standard mechanism for developing general context-aware systems. Nonetheless, the following have been identified as advantages for integrating context awareness in today's adaptable systems [7].

(i) Adapting the service presentation to service consumer devices, for instance, the images and videos that are suitable for the client's device.

(ii) It helps to adapt services dynamically to a new situation such as location, time, users profiles, networks and the device capabilities and

(iii)It could also help the service consumers to make decision on the requested service, if the system is able to provide context information to users.

Finally, considering context aware adaptable services would help improve the objectives for which the service has been created – improving usability of the service, the number of satisfied service consumers. However, this context information is complex to model. Location for instance is very complex and difficult to conceptualise. But some methods for modelling context make it easier to model any type of context data. These modelling approaches are briefly discussed as enumerated in [25].

(a) Logic Based Context Model

In this model, context data are expressed as facts in a rule based system. This enables automated inductive and deductive reasoning to be done on contextual information. The first order model allows an expressive description of context using Boolean operators and universal quantifiers. A good example of logic based context model is described in [31].

(b) Tagged encoding model

Context data are represented as tags with corresponding fields. It evolved from ContextML[60] which is an xml based protocol for transmitting contextual information

between a mobile client and a server.

(c) Key-Value Pairs

This is used to store context information as key that refers to environmental context variable and the value of the variable holding the actual context data. This type of context model uses pattern matching to query the context data and notify the adaptation mechanisms. An example of context model is used in Mobisaic project [61].

(d) Object Oriented Model

The context data is embedded in the states of the object and the object provides methods to access and modify the states as used in MADAM project [25].

(e) Ontology Based Model

The basic concept of this model is to provide vocabulary for representing knowledge about a domain and for describing specific situations in such domain. Ontology based context models define a common vocabulary to share context information in a pervasive computing domain and it includes machine-interpretable definitions of basic concepts in the domain and the relations that exist among them. An example of systems that use this model is SOCAM [65].

2.5 Dynamic Reconfiguration

A dynamic reconfiguration is one of the best techniques [52] to adapt a system as it helps to easily implement adaptation decisions. The idea here is to evolve incrementally from one configuration to another at runtime as opposed to design time while having little or no impact on the system execution. This traditionally takes place at runtime and it can be applied to rearrange various elements of various parts of the system such as application or services, platforms, systems architectures and management facilities. This means that systems need not be taken off-line, rebooted or restarted to accommodate the adaptation changes. Dynamic

reconfiguration however, requires information about the running systems. In addition, to reconfigure a system, necessary protocol must be defined. Context sensing with runtime monitoring can be adopted for this purpose. Such sensing must make provision for reasoning and evaluation of the sensed context. Triggering and realising reconfiguration should be based on some specified criteria and runtime monitoring of the system execution contexts. For instance, performance requirement may require the migration of some components so that they are closer to their source of demand. An alternative might be to host a particular service on a less loaded system so that the services can execute faster. Two major approaches in research have been proposed to achieving reconfiguration [22]. One is by adding configuration elements to the application modules which is not desirable approach as it makes the systems very complex and does not allow separation of concern. In such reconfiguration, the following elements maybe specified. One, the definition of interfaces of the systems modules in terms of provided services and required services. Two, localisation of the source files and three links between provided and required services of the application modules. This approach, however, offers a solution for structural changes by explicitly specifying components binding and by on-line replacement and duplicating of components.

In order to ease reconfiguration execution flow, reconfiguration has to be initiated at some points. This sequence consists of two steps: waiting to reach a reconfiguration point; and blocking communication channels to manage messages in transit while the component context is encoded and new components are created. Each of these approaches implemented a reconfiguration mechanism in the application codes which are not desirable as stated earlier on.

Another important way in which reconfiguration is effected is based on component and

configuration languages. This is different from the first approach explained above mainly because it supports various interaction schemas. In addition, components based reconfiguration provides additional run time flexibility coupled with separation of concerns it offers over the previous approach. For instance, the Architectural Description Language (ADL) can be used to create, validate and update architectures. ADLs are very useful in expressing component hierarchy and in specifying interaction, application deployment and the dynamic features of such applications. Some ADLs have been used to specify behaviour and non-functional properties such as performance, security, availability etc. by using additional interfaces to generate and execute code. ACME, C2, Olan, Aster, Rapid, Wright, Unicon [62,71] are few examples of such ADLs that allow the users to specify the behaviours of various entities of the systems and the components interconnections.

2.5.1 State of the Art of Component Reconfiguration

The component reconfiguration helps to easily implement dynamic adaptation and it is classified into two: Software and hardware reconfigurations [25]. To adapt the behaviour of a mobile device to prevailing context changes in its environment, application services running on top of such devices must be adapted by reconfiguration. In software reconfiguration, adaptation is achieved in two ways: by tuning the components parameters and by components composition. Component parameters tuning describes the processes in which a component is designed in a way that allows it to operate in more than one possible mode. Component composition is a more general and powerful way of adapting systems through reconfiguration [52]. This is because it allows applications to be reconfigured in a manner that was not anticipated at design time. In hardware reconfiguration, however, features such as display brightness and speaker's volume can be adjusted according to some context

changes. Also, the network adapter can be adjusted by switching them on or off or to a power safe mode. Furthermore, operating systems parameters can be tuned, for example the system storage and memory balance of a PDA. A significant characteristic of hardware reconfiguration is that the changes, typically, affect the whole device and consequently, all software running on top of it.

2.6 Related Work on Adaptable Systems

This section provides the analysis of some of the most influential researches in the field of dynamic software adaptation. The key areas of interest here include, but not limited to adaptation strategies as enumerated in section 2.2. These systems are analysed specifically with respect to their relevance to the aims and objectives of this dissertation. In this analysis, adaptable systems are classified based on their adaptation strategies, mechanisms and target platforms.

2.6.1 CARISMA

Context Aware Reflective Middleware System for Mobile Applications [66] is a research carried out at the University College London. It presents a design for peer-to-peer middleware based on service provision, whereby reflection is used to adapt the interaction between context-aware mobile applications. Each node of the system can export services and possible different behaviours or implementations for those services. Services can be selected according to user and application context information as specified in an application profile, which is an XML document embedded in the application. It is responsible to monitor the application execution contexts especially by querying the underlying network-enabled operating systems. An application can request to view and modify their profiles at runtime, thereby adapting the middleware as application specific and user specific requirements

change dynamically. It also provides the ability for the application to be informed by the middleware of specific execution conditions, supporting the development of resource-aware applications. This system is specifically based on the provision of multiple implementations of the same service with different behaviours. This concept is similar to the concept of using variants of a given service to adapt it to current context situation. These variants are various implementations or functions of same service but with varying quality of service.

Unlike the work in this dissertation which adapts the interaction of the service to the execution context changes to ultimately reduce response time, CARISMA is specifically built to adapt the service to user's contexts. Not only this, it primarily focuses on the identification and resolution of profile conflicts and not on the actual provision of service adaptation to other context situations. CARISMA adapts service behaviours in response to context of execution using reflection. In the process of adaptation, some aspects of the application are altered. This, however, is not desirable as we want a situation whereby the service consumer though may not have exactly the requested service, but would be satisfied with what she is presented with by reducing the service response time.

2.6.2 M3

The M3 [75] architecture is an adaptable middleware framework that supports adaptation using context-awareness. This it achieves by using a mobile Enterprise Architecture Description Language (MEADL) script to dynamically reconfigure how application components interact with each other within runtime environment. In this system, all components interact and coordinate with each other using events. As these events occur, they are monitored and used to trigger adaptation of the architecture and the underlying collection of distributed services and network protocols. The M3 runtime environment maintains

context variables that can be used to perform adaptations of this application in a context aware manner. While our work has many similar design ideas to this project, the M3 system has some important drawbacks. The adaptation mechanisms prototyped (including filtering, object migration, interface restrictions and web content adaptation) all lack the generality and openness of a general-purpose adaptation mechanism as used in our design. Morphable objects, i.e., objects that can change their type at runtime, are mentioned in [75] but no more information is available about these reflective techniques. However, our design adopts the utility function for general purpose dynamic adaptation decision making.

2.6.3 MADAM

In the Mobility and Adaptation Enabling Middleware approach, applications are built as components frameworks. Explicit models of the application framework architecture that specify the variability are used by the MADAM Middleware to reason about and control adaptations at runtime. The central adaptation control loop of the middleware detects changes in the environment, reasons about and decides on suitable adaptations to fit the new operating conditions. and then implements the adaptations through reconfiguration of the running applications. To enable the middleware to distinguish between implementation choices at variation points, components are annotated with property evaluator functions which may represent QoS characteristics. The decision on which adaptation to make is done by the MADAM planning framework. The planning activity consists of dynamic discovery of implementation alternatives at the variation points of the application component framework, and the selection of those variants that match both the current user and device contexts.

Adaptation alternatives evaluations are performed by a utility function which composes and

weights result from the property evaluator functions of the component. MADAM planning framework is domain independent and has been applied to many adaptation types. The middleware is built as an open framework and each of its components dealing with adaptation management, context management, and dynamic reconfiguration and deployment are separated from the core. This openness makes MADAM useful for the kind of adaptation this work addresses as its open frameworks can be adapted to effect our adaptation strategies. This helps to reduce the prototyping time and eliminate re-inventing the wheel. However, MADAM does not address adaptation from service oriented point of view and does not assume a grid environment in its design.

2.6.4 Sparkle

The Sparkle Project aims to provide an Internet-enabled infrastructure for mobile computing which supports dynamic component composition. Applications are seen as a means by which users perform tasks [38]. An application usually provides several functionalities which users can invoke to fulfill their tasks. According to Sparkle, applications can be broken up into components along the lines of functionalities. Every component provides certain functionality. There may be more than one component which fulfills the same functionality, at runtime, the appropriate component is brought in and executed. Hence, applications are linked by functionalities, rather than by exact components. Consequently, when one runs an application in different environments, to carry out the same task, the actual components used may be different. In this work, clients send requests for the components to the network, and are returned with the appropriate component. Since applications are linked by functionality rather than by exact components, the requests specify functionality requirements rather than component identifiers. They also include non-functional requirements such as run-time

resource information and context, so as to determine which component would be most suitable for the client. From the client's viewpoint, there is a lot of reliance on the network. The network stores the components and also possesses intelligence to match the appropriate component for the client. This project uses component model similar to our component model for adaptation as application are neatly arranged into component which can be composed dynamically at runtime. These components are predefined and are stored in the network. If the network is not available or the network condition become worse, then applications are not able to run. It also uses resource managers that maintain information about physical resources, contexts and connectivity. This is similar to our context monitor that maintains similar context and resource information. It, however, uses proxies for matching adaptation whereas we used the evaluator that uses planning and utility function for adaptation decision.

2.6.5 Aura

This project looks into the issues that affect component-based development of pervasive systems [49]. It considers pervasive systems as collections of cooperating components that achieve users' tasks. Aura project proves that software systems in a pervasive environment must exhibit the following characteristics for them to be adaptable in such environment. One, they must exhibit mobility. This means that tasks must follow users as they move from one device to another. Two, they must exhibit adaptability. Tasks can take advantage of resources as they change and three; they must be resource-aware. This means that components must publish their resource requirements and offer multi-fidelity of services. Aura models applications in terms of tasks. Our model is very similar to this project in the sense that, our model must be able to integrate the resource and context requirements of each running

service for adaptation decision making process.

2.6.6 DACIA

This project provides a framework for building adaptive distributed applications. In DACIA, applications are made of components located in various network entities and the links between components represent the direction of data flow within the application [76]. It considers components as processing and routing units, transforming one or more input data streams. There are monitoring modules specific to each application, which are responsible for monitoring application performance and making configuration decisions. These applications are adapted dynamically by adjusting the connection between components and location of different components, and hence leading to a change in the applications. This appears to be a good framework suitable for distributed applications which require data flow from one entity to another such as video-on-demand applications however; the adaptation in DACIA is application specific meaning that every application implements its own adaptation policy. This makes the adaptation very static and complex to develop. This also constitutes a lot of overheads in terms of resources requirements. In our design, we look into a more dynamic and general adaptation policy that is not application specific, but instead optimises required service execution resources. In this case, the system has a wider view of the resource needs of all running services. It will also help make services run faster, now that services are not saddled with the responsibility of deciding environmental resource availability.

2.6.7 Globus

This is a framework that allows for both resource reservation and application adaptation. In this architecture, heterogeneous resources are modelled by *resource objects*. The architecture

of Globus[15] encompasses several components. The *information service* provides information about resource properties and resource availability. A *co-reservation agent* is in charge of both mapping application QoS requirements to specific resource requirements and requesting the reservation of such resources. The task of a *co-allocation agent* is similar but focuses on the allocation of resources. *Local resource managers* are responsible for attending request for allocation of resources. Adaptation is then achieved by the use of three distinct mechanisms. Firstly, *sensors* are in charge of monitoring both resources and the application behaviour. Secondly, *decision procedures* allow for the selection of an adaptation strategy. Lastly, *actuators* are the means to dynamically modify both resource allocations and application behaviour. This pattern of design helps us in the design of the adaptation model. The Globus project can form our Grid end infrastructure upon which adaptation service module is built. It will provide the resource management for Grid infrastructure. But the design of Globus does not consider mobile service and we hope that the combination of the resource rich Globus platform will help make up for the resource poverty of mobile devices.

2.6.8 GRACE Project

In its approach [63], system layers in GRACE are designed with the ability to *adapt* in response to system or application changes. Further, to achieve the full benefits of these adaptations, all system layers *cooperate* with each other to determine a system-wide globally optimal configuration. For example, for real-time video delivery, the simple choice of the compression technique entails a trade-off between computation time, energy and number of bits (bandwidth). For a globally optimal solution, the choices for error correction for the unstable wireless network and protocols for congestion on the wired part of the network were considered. The presence of multiple applications contending for the same resources and the

ability of the processor to adapt its performance/energy tradeoffs, further reduce the possibility that any layer can by itself determine a globally optimal solution. The centre piece of the GRACE project is a cross-layer adaptation framework that enables coordination of the adaptations at the different system layers, for the best QoS possible. The key challenge lies in exposing only relevant information across layers without compromising the current advantages of having system layers that are virtually closed to each other (i.e. selective transparency).

Thus, the solution has the following properties: (1) It performs its global cross-layer optimisations without exposing implementation internals of a layer to other layers, (2) It localises adaptation decisions specific to a layer, to within that layer, (3) A system component that exploits adaptation capabilities in other system layers must also be usable with implementations of those layers that do not have the same adaptation capabilities. However, the focus of this project has been on energy management for single multimedia node. The adaptation we designed is aimed at generic services meaning that it can be customised to handle any type of service adaptation. But understanding of the use utility to model adaptation in GRACE project has been very helpful.

2.6.9 ACCORD

Administering Connected Co-operative Residential Domains (ACCORD) uses a closely related approach to the one used by our model as it helps to develop compassable adaptable applications. It manages behaviours and compositional (interaction, organisation and coordination between components) aspects of an application using high level rules, injected at runtime, and enforced by an agent infrastructure. Instead, our work relies on planning [50] to ensure correct execution of reconfiguration actions. In ACCORD, adaptation element is

augmented with Element Manager that monitors its execution contexts, and fires adaptation rules. On the other hand, our model uses utility based policies rather than rule-based policies to fire adaptation actions. It also separates the monitoring module, the dynamic evaluation model and reconfiguration module. This means that services can now be adapted with ease by service developers, making services to be easily managed, debugged, and maintained.

2.6.10 SECAS

Simple Environment for Context Aware Systems described in [7] is a platform that attempted to make services, data and user interface adaptable to varying contexts in a mobile environment using standard methods and approaches for their solution. It is based on four sub-systems: the application core, the adaptation layer, the context management and the client side. A set of components for each of these subsystems manages the operation needed for sensing and interpreting the contexts and for adapting consequently the application core and the user interface. SECAS's context management is highly dependent on the environment and on the sensors that capture and transmit raw context data. The complexity of designing such context sensors and interpreters is encapsulated in context Providers and context Interpreter. The context providers are responsible for managing context aspects such as user profiles and preferences. The interpreter translates the low level context into high level more meaningful representation. Some part of this context is dynamic and volatile meaning that they are consumed as soon as they are sensed while some are static meaning that they are saved a repository. Such contexts usually do not change frequently. The adaptation is applied to application services, exchanged data with user visualisation. This means that SECAS uses content and user interface adaptation strategies. However, details of how their adapter selects a feasible service instantiations were not given. A lot is learnt from

this work as its modular design of context management, and its idea of having various instantiations of a given service are very useful for our design. However, the ideas of service reconfiguration and utility modelling were not used which we believe has an edge over rule-based model for dynamic adaptation modelling.

2.7 Concluding Remarks

This chapter has described a number of systems. The summary of the system discussed is given in table 2.1. It has also discussed research that influences the goal and objectives of this dissertation. The main objective of this research, as stated in chapter one is to design adaptation mechanisms and algorithms in order to reduce response time between service and service delivery in CACIP executing environment. Others were to develop a conceptual adaptation model and dynamic decision making algorithm for the model. In order to reach these objectives, a number of requirements must be met. An adaptation model that supports interaction of service components in order to reduce response time is needed. This adaptation must be able to perform adaptations on service components as they are composed to form these services. To test the model, an adaptation framework is needed that supports the dynamic context monitoring, evaluation, and reconfiguration of service components at runtime. From the research described, it can be seen that there currently exists no mechanism that completely fulfils the objectives and the dissertation continues next chapter with detail design of CAAM model that meets objectives and requirements of this research. For instance, the issue of adapting interaction between service consumers and providers were not properly addressed. Also, some of the adaptation mechanisms and policies used in those systems were static and are not good for dynamic adaptation process. Therefore, a Context Aware Adaptation Model (CAAM) is proposed to address these gaps.

Table 2.1: Summary of Reviewed Adaptable Systems

Adaptable Systems	Adaptation Strategies Explored	Adaptation Mechanism	Context Awareness	Reconfigurations	Adaptation Policy	Application platform
Accord	It uses condition objects to measure state of resources and services	Uses rule-based and agent mechanism for adaptation	No Context awareness but is aware of execution resources	No reconfiguration, lacks capability for dynamic component composition	Uses contract-based policy	For stable systems. No grid or mobile applications are assumed
M3	Adapt Interaction of application components	Uses events, network protocols as contexts to adapt applications interaction	Not aware of user contexts	Uses reconfiguration described by a MEADL scripting language	Rule-based policy adopted	Targeted mobile applications
RAM	Adapt application roles	Uses reflection and meta types	Aware of application contexts	No reconfiguration	Uses rule-based policy	Mobile applications targeted
DACIA	Adapts component connection	Uses application performance	Aware of application contexts	Reconfiguration used	Uses rule-based policy	Not targeted at mobile application but distributed application
AURA	Adapts application's task which is service with QoS specifications	Uses tasks as for adapting the system to context changes	Aware of device, user and application contexts	Uses Reconfiguration to effect adaptation decision	Uses utility function-based policy	Targeted mobile applications
CARISMA	Adapt application interaction and behaviours	Uses Reflection to monitor and adapt interaction	Aware of applications and user contexts	No reconfiguration	Uses rule-based policies which are embedded in	Mobile applications targeted

		among mobile applications			the applications	
SECAS	Adapting user interface and data to changes in contexts	Based on web services	Context-aware	No Reconfiguration	Not available	Mobile services are targeted.
Globus	Uses, reflection and meta types to adapt software dynamically at runtime	Not-Context aware but it's resource aware. It does not consider the context of the users and devices.	Rule-based policy	No reconfiguration	Uses decision procedures for adaptation policy	Mobile services and devices not considered.
GRACE	Uses the idea of crosslayer adaptation and utility to quantify the quality of application configuration	Not-context aware	Uses utility function model for decision making	No reconfiguration	Uses utility function model to quantify the quality of application configuration	For single node multimedia application
MADAM	Compositional, parameter adaptation adopted to adapt software components and devices	Context aware, resource aware	Utility based adaptation policy	Uses component reconfiguration	Uses utility function and planning for dynamic adaptation decision	Designed only for mobile applications Service interaction and grid services providers are not considered

CAAM	Compositional, parameter adaptation,	Context aware, resource-aware	Utility-base adaptation policy with on the fly decision algorithm	Use s service reconfiguration pattern for executing adaptation	Uses utility function and planning for evaluation service contexts and for adaptation decision	Takes care of service consumer and service provider interaction in Grid Infrastructure

CHAPTER THREE

MODEL DESIGN AND DEVELOPMENT

3.1 Introduction

This chapter describes the design of the dynamic adaptation model. The chapter begins with the aims and objectives of this project followed by an example scenario as presented by the author in [47]. Thereafter, some important requirements for the dynamic adaptation model are identified. The chapter then continues with an in-depth discussion of a proposed context-aware and reconfigurable model, explains some of the concepts adopted in the design, and presented how the model can be used for adapting service interaction.

3.2 Requirements for Service Interactions

In chapter one, it was stated that one of the objectives of this investigation is to design suitable adaptation mechanisms and algorithm for CACIP that reduces response time between service consumers and grid service providers. Others are to develop conceptual model for CACIP service components interaction adaptation, to design a dynamic adaptation decision making algorithm and to demonstrate the design and model. Keeping these objectives in mind, a number of requirements that meet the needs of the adaptation model are unveiled by illustrating a simple scenario from which such requirements are generated.

3.2.1 Example Scenario

To identify the requirement of the model, a scenario is presented that is typical of such a ubiquitous environment that the envisaged model fits as follows. Eunice has just arrived in South Africa coming to Europe. On arrival at OR Tambo International Airport, and while

waiting for immigration clearance, she switches her blackberry device on and connects through an enterprise wireless LAN at the airport. After necessary authentication and authorisation, she is immediately provided with a number of services available in a Grid environment. She plugs in and requests for mobile entertainment service. She is specifically interested in a recently advertised home video which she has longed to watch while in Europe.

However, a call comes telling her that her train will soon leave for the country side where she is scheduled to meet with her parents. As she rushes to board the train, she decides to use the cell phone function of her blackberry device while she pauses the video service until when onboard. The service is then transferred to her mobile phone as she connects using GPRS network and adapted to that terminal. She, however, changes her mind to watch a live broadcast of the president address to the nation which was announced but noticed a serious delay in response to her request as the train fast moves by. She then requests to connect through a 4G network, which is higher in price compared to the GPRS network. At this point she notices that her phone battery level has gone so low and hence requests to watch the broadcast in black and white video mode. In this mode, her phone consumes less energy. From the above simple scenario, some requirements for designing adaptation for such service provisioning environment can be identified.

It could be observed from the scenario that we need systems that can continuously configure and reconfigure themselves under varying and unpredictable conditions. They should equally be able to dynamically monitor and tune the resources available to them. It is believed that building adaptive behaviour into mobile systems will enable them to respond better and adapt to changes occurring around them. In order to achieve these, the following seven

requirements have been identified.

1 Security: In a distributed system such as Grid, entities interact with some other entities with which they may not have complete trust. There is therefore need for authorization and trust management. The decision about this sometimes has to be made in the presence of strong existing trust systems. The classic Grid has very strong secure authentication but not in a transient, volatile and unstable mobile Grid environment.

2 Extension of Functionality: This is needed to automatically extend systems functionality so that it could be discovered on the fly. This could be to discover new hardware resources or discovering recent dynamically created services.

3 Availability of Networks: Eunice, in the above scenario, has the option of switching among networks if the capacity of her existing network could not help deliver her needed services. She can switch between 2.5G, 3G, 4G, Bluetooth, or even WLAN. She should also be able to switch between both on-line and off-line modes.

4 Service Presentations: The presentation of requested services needs to adapt to the interfaces of the devices used for this purpose. For instance, switching from laptop to mobile devices should not hinder her from having access to the service and have same level of satisfaction with the presentation.

5 Service Redeployment: The services should have the capability to be redeployed on various devices in order to improve their efficiency.

6 Changing the Quality of Data: There is need for the system to be able to vary the qualities of the data based on context changes. For instance, when Eunice listens to the president's address, and suddenly there is a drop in the network bandwidth, changing the video compression mode might be affected to preserve the quality.

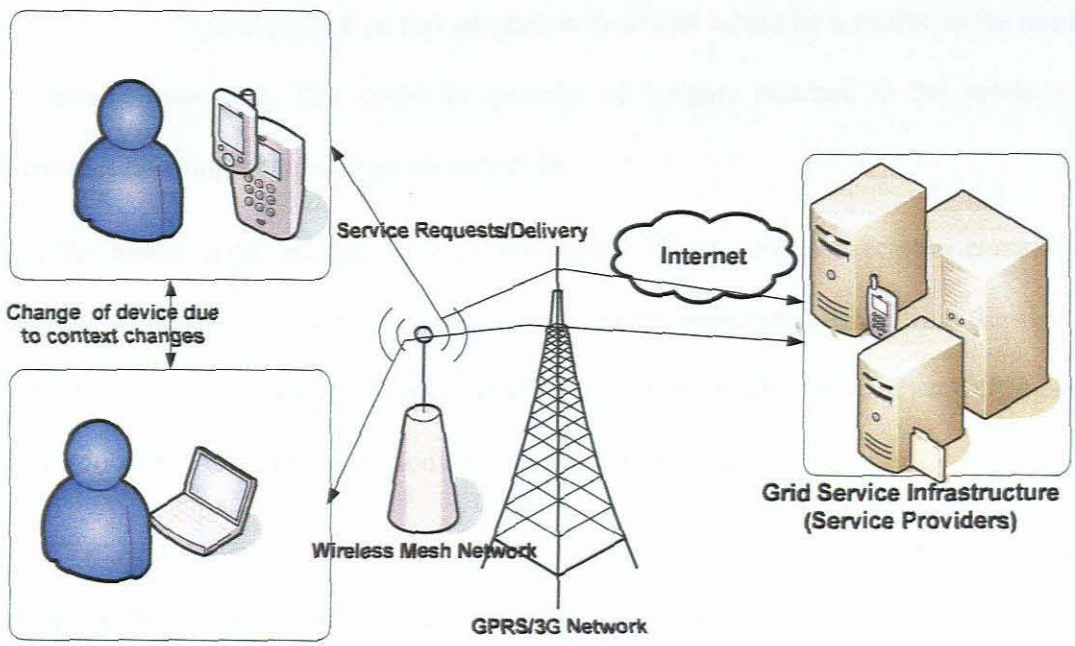

Figure 3.1: Service Adaptation Architecture

7 Context-aware Interaction: The system should be sensitive to the interaction taking place between the service consumers and the service providers. This is necessary for the systems to be able to take both reactive and proactive decisions in case there is a delay longer than necessary for the service delivery due to context variation.

These are some of the basic considerations in designing models for the type of adaptation this study addresses. However, among all the stated considerations, this work focuses on investigating adaptable context-aware interaction and Figure 3.1 illustrates such environment.

In order to achieve this, the following requirements guide the design of the adaptation model:

(1) The service consumers must be able to specify their quality of service preferences in a

flexible way. This is necessary so that adaptation decisions would be sensitive to the needs of the service consumers. This could be in form of weights attached to the services that determines how important such given service is.

(2) The model needs as input the current status of the service execution context. The execution context of both service providers and service consumers change from time to time. In order to increase the agility of the adaptation model, it must be aware of such changes and use this to either reactively or proactively configure adaptation.

For instance, the consumer's needs change from time to time, the network connecting the devices is very unstable, and the devices themselves are constrained by their fluctuating resources.

(3) The model must have the option to accept or reject new service request or adapt to the prevailing context situation. This is necessary to make the users not endlessly wait for a service that will never be delivered.

(4) The model must make provision for dynamic reasoning on the context information it acquires from the clients, environment and users. This requirement is one of the objectives of this investigation as adaptation decision making is still an active parts of mobile Grid adaptation researches [25]. There is need for the systems to make dynamic decision over the context changes that occur in the system runtime environment.

(5) The model needs to provide some quality of service by reducing the delay experienced due to contexts changes. Ultimately, the need to adapt the interaction of service clients with service providers is to improve on the quality of service.

The next section presents the interaction model of our envisaged systems, this presents how service consumers and service providers interact to get services delivered.

3.2.2 The CACIP Interaction Model

Having identified the requirements for the design of the adaptation model in the last section, now it is time to discuss how CACIP interaction architecture fits into mobile Grid context being one of the objectives of this dissertation.

3.2.2.1 CACIP Interaction Bus

Recent years have witnessed explosions of software infrastructures such as middleware platforms that have dominated distributed applications development with grid systems [4] as good examples. One of the goals of Mobile Grid computing is to access computational resources automatically on demand to deliver the services required with appropriate quality of service [3]. Obviously, mobile devices are now increasingly common, therefore, an infrastructure is required that allows mobile devices to use Grid services, consequently enabling the execution of complex, resource-intensive applications on the resource-constrained devices [4].

However, because of the resource limitations of mobile devices, it becomes complex and challenging to build applications that run on those devices. We share the view that the marriage of mobile devices and grid services will greatly minimise the challenges of mobile devices interaction with grid service providers and at the same time making grid services accessible anytime anywhere [58]. Typically, design paradigms partition these infrastructures into components that can be reused, and interconnect them through constructs provided by such interaction infrastructure. However, service interaction requirements vary. Some gaps, therefore, do exist between the interaction that such software service provides and those required for interconnecting application service components. One way of bridging these gaps

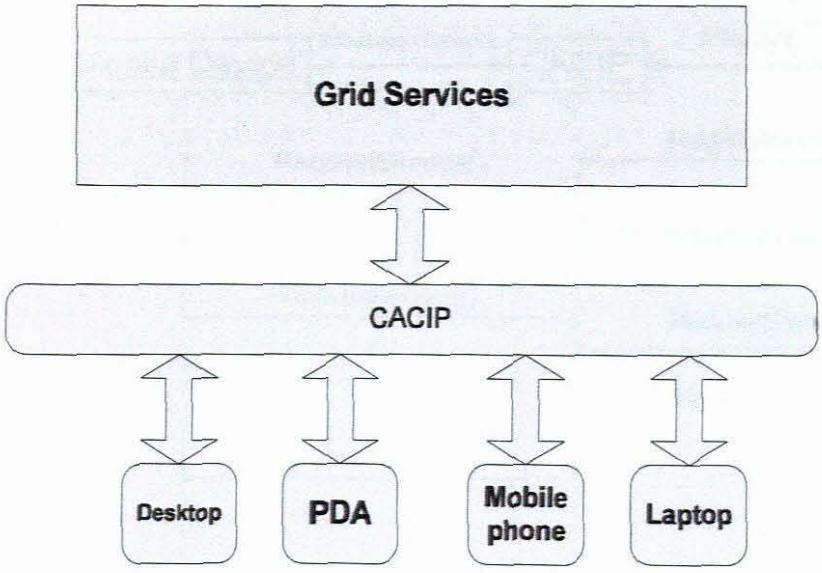

Figure 3.2: An Architecture Showing Mobile Devices Interacting with Grid Service

is to argue for explicit design of interaction systems between service consumers and service providers in such distributed systems. The classical clients/server paradigm of interaction is not suitable for mobile and wireless interaction because it assumes persistent, fixed and stable network link among communicating entities. Mobile systems, however, are prone to unanticipated disconnection due to environment fluctuations. Mobile applications that are designed to access Grid services reside in various mobile devices distributed over the wireless network. CACIP (Context Aware Component Interface Pattern) [5] adopts information bus as a persistent data structure that facilitates asynchronous communication among service components. These components rather than interact directly, register their messages in form of tuples deposited in the bus. The messages are pre-fetched and kept in the bus in anticipation of mobile client's request for it. It also helps in routing services among participating components in the interaction. This means that services are made available to the bus by service providers and consumed or retrieved by service consumers. Figure 3.2

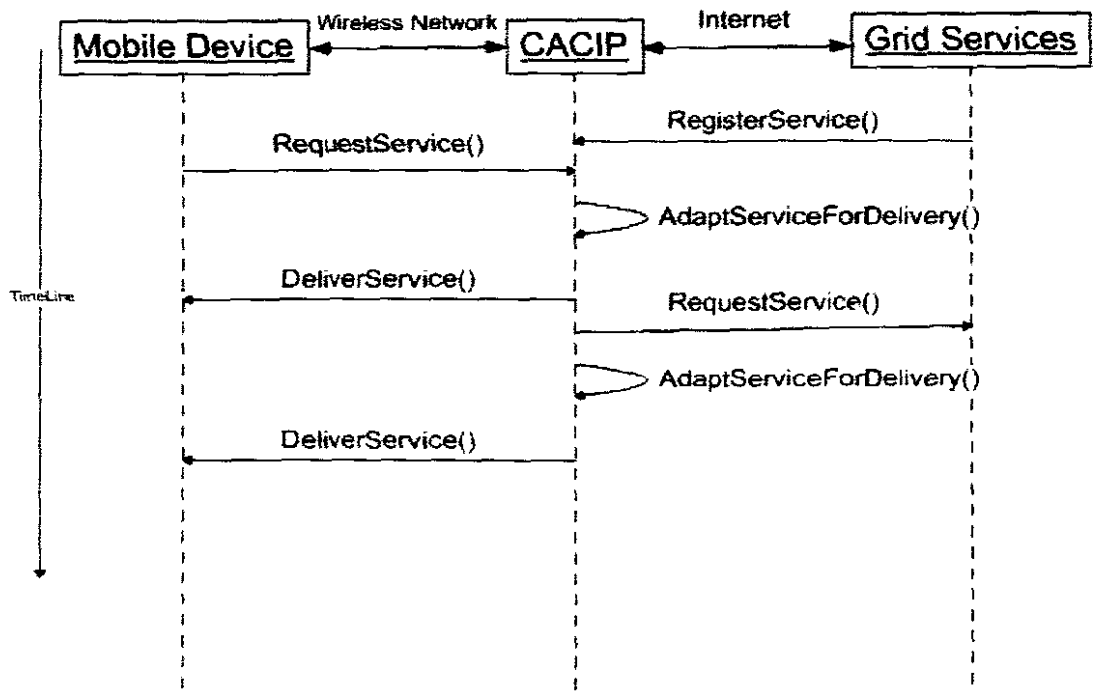

Figure 3.3: A CACIP Model of Grid Service Interaction

illustrates the new CACIP Architecture. One interesting advantage of this bus, upon which the adaptation model is built, is that components can connect and disconnect from it without affecting other components as the presence or absence of a particular service component does not depend on that of other components. The participating clients or service consumer's first need to register with the bus specifying the type of services it is interested in, while the service providers publish services they offer. As depicted in Figure 3.3, there are mobile devices as service consumers or clients, CACIP bus as service registry, and Grid services as service providers. These mobile devices are usually portable but are resource-limited with varying network capabilities. The CACIP bus might be running on a server or a group of servers located on a LAN and maybe accessible to the mobile devices via wireless network. The CACIP bus is also connected to the Grid via a high speed network. Different types of

Grid services are registered with the CACIP bus, providing convenient mechanism for mobile devices to find needed grid services.

The requests from mobile devices are sent directly to the CACIP bus. If the requested service is available, CACIP processes, adapts and sends it to the service consumers. Otherwise, if the bus cannot locate the service, it then invokes a request to the Grid service. Once the requested grid service is deployed and contexts of the service are good for its execution, then the adaptation model processes and adapts the service before it is sent to the service requestor or consumer. Figure 3.4 is a bird's eye view of the adaptation process. The CACIP bus interaction protocol assumes that there is a strong trust relationship between the Grid service and the CACIP bus so that the bus can interact securely and easily with the Grid. Therefore, this matter is not discussed any further in this dissertation.

3.2.2.2 The Service Interaction Definition

Having presented the CACIP bus interaction model, what is next in line is a methodical formulation of the service interaction problem as this basis of the adaptation model proposed in this dissertation.

Interaction between CACIP clients and provider components is therefore defined with a function:

$$N = f(R) \qquad (1)$$

where R and N represent service request and response respectively.

Assuming $R = \{ r_1, r2, r3, \ldots, rm \}$ and $N = \{n_1, n_2, n_3, nm\}$ which are vectors of typed values.

Each notification sent by a component is an object with a name and a return type. This representation offers a standard exchange format between components that facilitates composition and adaptation of interaction between clients and service containers. However,

these entities have functional dependencies that can prevent service delivery on timely basis since a request from a client may depend on the notification from another service component. To define these dependencies, we use the relation (f_1, f_2... f_n) → f, f (there could be many of it) represents a service consumer and f_1, f_2... f_n represent service providers as illustrated in Figure 3.5. This means that for service consumer f to successfully interact and access services in components f_1, f_2,...,f_n, some specific contexts of the device, the network and user must be satisfied. This dependency may be illustrated by a graph where nodes on the graph represent the components and the edges are the conditions that allow the components to interact with each other.

Now, based on Chaari et al [7], a general model of the envisaged mobile Grid application is defined as a tuple:

$M = (S, V, T)$ (2)

Satisfying the following essential requirements: S is a finite set of Service providers and consumers $\{S_1, S_2,...,S_n\}$, $V = \{v_{11},v_{12},...,v_{ij}\}$ is the set of service variants provided. And these are alternative implementations of a given service. T is a finite set of interactions $\{t_1, t_2... t_m\}$. Each t_i is a tuple (D, C, A) where D is the maximum delay or latency for interaction, C is the general context of execution of the application, and A is a finite set of

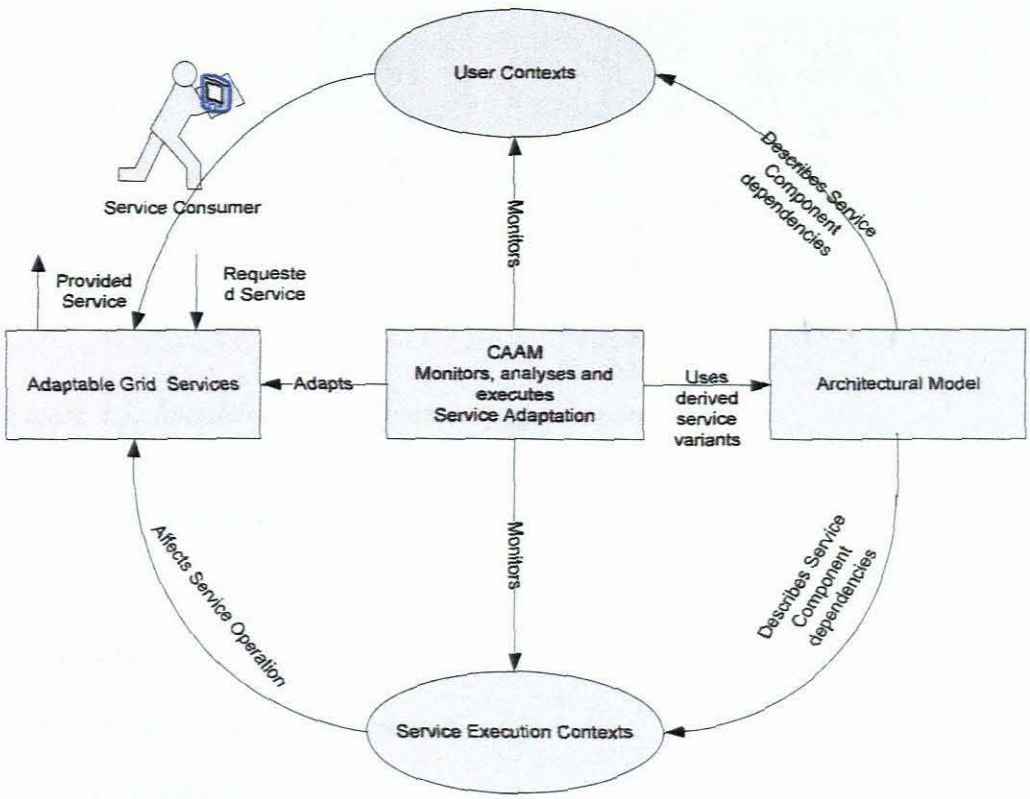

Figure 3. 4: Context Aware Adaptation Model (CAAM) Adaptation Process

dependencies among the components. Each dependency is a finite set of pairs (Required Context and Available Context) defined for each actor, for example, the service consumers and providers in the CACIP bus. Requested service will be executed if the contexts of execution satisfy some constraints for such interaction otherwise, the system need to effect adaptation. This is shown in figures 3.4 and 3.5.

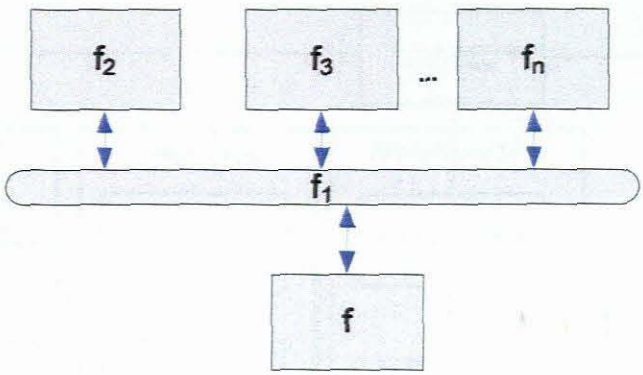

Figure 3.5: Interaction Dependency among Components

C is a set of constraints specified in the realm of the source context and the destination contexts for successful service delivery. With this definition, the need for adaptation of the interaction using context awareness has been established. Next, the architecture of the model to address this issue is then presented.

3.3 The Adaptation Model

The previous section defined CACIP interaction problem and established the need to design an adaptation model for handling interaction between CACIP entities. In this section, the model is presented with the strategies employed to address service interaction problem in a mobile grid environment which causes prolonged service response time. The model combines Service and reconfiguration pattern [77, 79] and context awareness to achieve its goal. Every component of this architecture works in harmony in order to achieve the ultimate goal of the proposed utility function-based context aware adaptation model.

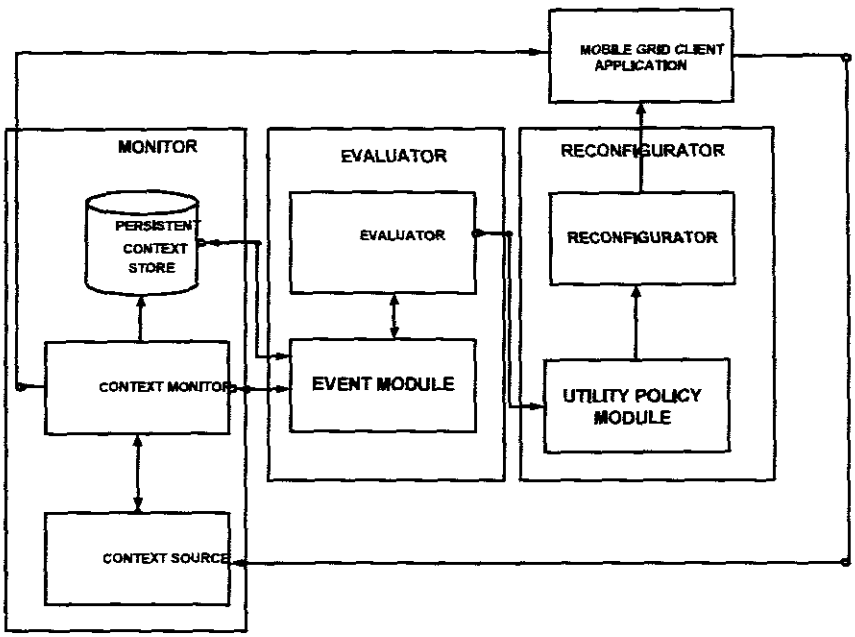

Figure 3.6: Architectural View of the Context-Aware Adaptation Model

The Context Aware Adaptation Model (CAAM) model architecture is depicted in Figure 3.6. It consists of four main subsystems namely: The Grid service consumers, the *Context Monitor*, the *Context Evaluator*, and the *Service Reconfigurator*. Since this work is to adapt the interaction of mobile Grid clients to changing environmental context of mobile terminals, wireless network and that of Grid services, the overall goal is to improve the service quality experienced by service consumers in terms of service response.

3.3.1 Context Monitoring and Event Management

This section discusses the design of the monitor component with a view to communicating monitored changes among other components of the model.

3.3.1.1 The Monitor

The monitor observes the changes occurring in predefined context parameters. Here, context refers to the states, resources and other conditions that affect the execution of

services. For the system to adapt in response to these context changes, it is necessary that the context be monitored to track such context changes. Two main methods for Context monitoring are identified [27]: the event driven and time driven methods. The time driven approach is about periodic probing or polling of the system being monitored to provide a view of the context status and suffers from missing some contextual event changes between when it was last probed and present probes particularly if the sample rate is very low.

However, the event driven approach brings liveliness into monitoring without any need for periodic probing. The event driven approach waits for a change in the status of the context being monitored to occur and then sends a notification of such changes. In this design, the event driven approach is adopted. Since the time driven approach comes with a lot of overheads and trade-offs between time spent probing or polling for change in the context values that may not have occurred anyway. The state or status of any object in this case, contexts, is the representation of the cumulative results of its behaviour and this is represented by the values of all its properties. These events are atomic entities that reflect a change in the status of the contexts being monitored. It is worth noting that this status changes continuously, therefore, these changes are observed as subsets of those events that are of significance to adaptation decisions.

We, therefore, define the events in this design as those dynamic adaptation triggering policies that help decide on what CAAM does should adaptation be necessary. Whenever these events are fired, any policy or rules associated with such events is executed. Two or more events can be used to execute a triggering policy. For instance, the *BandWidthLow* and *DeviceMemoryLow* events maybe used to carry out execution of a given service in another device with enough memory and network connection bandwidth. Therefore an

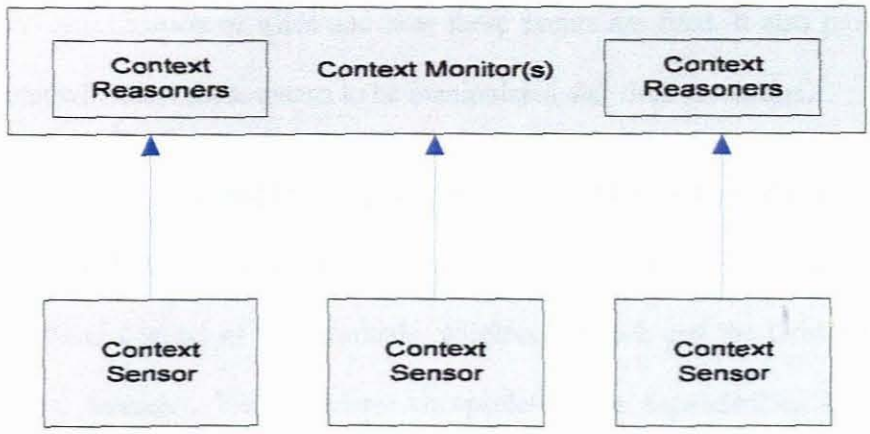

Figure 3. 7:Context Monitor Architecture

event triggering maybe defined as:

If *(BandwidthUtilityLow* ***AND*** *DeviceMemoryUtilityLow)*

Execute *Action1*

Where ***Action1*** may represent the execution of such service on a remote device with enough memory and network bandwidth.

Different combinations of such events can be associated with the execution of various adaptation policies. This means that this event makes it possible to have specification of composite events thereby making provision for event combination and manipulation. This idea is very central to the design of the monitoring and evaluator modules of the adaptation model. One of the design requirements for the monitoring module is that at design time, no general assumption on how to anticipate context changes is made. This specification of what constitutes interesting contextual changes is dynamically done at runtime. Another consideration in this design is that the system is capable of dynamic definition of events and dynamic specification of conditions that will fire or trigger events. The event management module is, therefore, responsible for dynamic definition and registration of new events and

the specification of when and how these events are fired. It also provides the mechanism that will allow these events to be manipulated and fired at runtime.

The event management supports the entire operation of the adaptation event driven communication. The context monitor is highly dependent on the environment which is the physical context of the terminals, wireless network and the Grid services as well as the event manager. We, therefore, encapsulate such dependencies in ***context sensors*** and ***context reasoners.*** The *context sensors* and *reasoners* are responsible for retrieving and supplying the context data to the context monitor through the event management module. The context sensors are responsible for observing and gathering context information from context sources. The context sensors are components of the model that provide context information to the persistent context store through the context monitor or directly to the Evaluator. They also provide context information to ***context reasoners*** that perform operations on the context data such as abstraction, aggregation, derivation, and prediction from primitive context data. They are primarily used to derive new context information from original context data. This is the case of the persistent context store where the ***context reasoners*** take context data from the ***context store*** in order to provide new context information on the trends of context changes in the system. Figure 3.7 gives a clear picture of this concept.

The context Evaluator utilises some of these parameter values directly as soon as they are produced by the Context monitor. Those that are not consumed immediately are stored temporarily in the ***persistent context store*** so that they can be retrieved later by the ***Evaluator*** through the asynchronous event management module for proactive adaptation

decision.

3.3.1.2 Context Modelling

The following context types are important in providing adaptable services in the context of this research.

(a) **Terminal device contexts**: The adaptation mechanism must be aware of the capabilities of devices being used to request services. Capabilities such as device display size, memory, communication interfaces, available multimedia, processing power, and encoder/decoder are good examples.

(b) **User Context**: This will include things like user subscriptions relating to types of services being requested for, consumer's identity, access right policies, and privacy aspects for managing and exchanging stored context preferred by context consumers. However, in this design, this is not really the focus of our context design as this was taken care of in the previous design of CACIP.

(c) **Service Environment Context:** This includes the consumer environment such as time, location, light, noise, activity and mobile clients' wireless network; such as network latency, bandwidth, network availability are examples.

3.3.1.2 Context Model

The context model refers to the format in which the context data are abstracted. Contextual information needs to support very broad adaptation decisions in different combinations. However, to handle such context data that is provided by the context sensors, this data should be structured in such a way that it will be easy to reason about them. A context-aware system requires context information to be exchanged and used between different entities such as

users and services in the same semantic understanding [17]. This means that an appropriate context model should support semantic interoperability that enables the common schemas to be shared between different entities. In the scenario illustrated in section 3.1.1, Eunice location must be understood by her blackberry device. Also, context data can have variety of characteristics. The definition of context, therefore, includes information in any given domain. This context information is interrelated, for example, available bandwidth and available memory may both determine what adaptation strategies to effect. Finally, this context information is not static but dynamic meaning that they are not consistent as they change from time to time. Table 3.1 is a summary of example context elements this model considered.

Different ways and means have been proposed to model such context information as illustrated in chapter two. However, among these models, we decided to model our context using the key-value pair as used in Mobisaic [80] technique that uses the key to refer to the context variable and the value of the variable holding the actual context data. The decision to choose this model type is informed by the fact that this is for now the best model that provide us with simple context abstraction with such values that can be consumed by the our policy model. In most cases such context model uses pattern-matching queries to notify adaptation mechanisms.The context model is illustrated in Figure 3.8. Entity represents any context that is relevant for the purpose of execution of any given service. In the entity representation, each of our context types is also modelled.

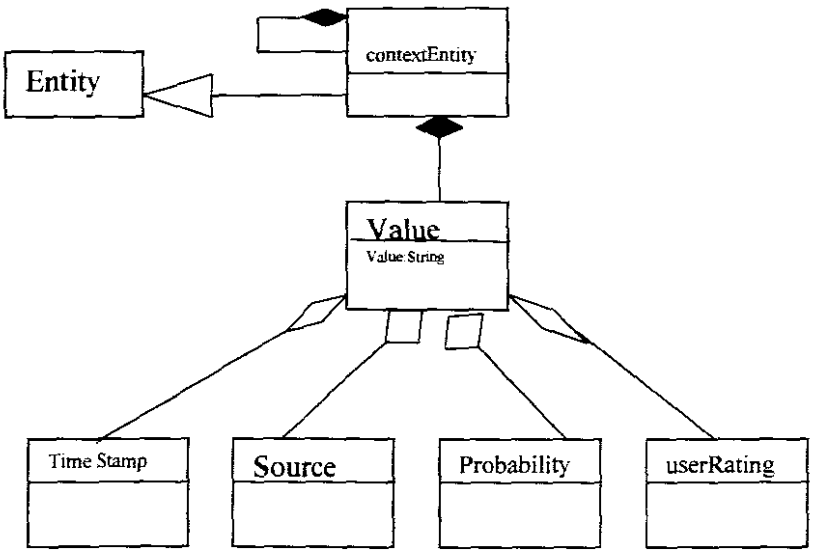

Figure 3.8: Context Model Adapted from MADAM [25]

The ***userEntity*** represents entities of interest in the user environment. Example is user location, light, sound, etc. ***ResourceEntities*** represents a runtime source of supply. A resource entity has limited capacity and these are represented by property types. Resources such as network bandwidths, network types such as GSM, GPRS or 3G, network channels, device memory, device CPU power etc are good examples of this entity type. ***ServiceEntity*** represents service based entities.

Service entities may be composed of other service entities and use other service entities or resource entities. A context entity encompasses the context information as illustrated in figure 3.8. This can be composite, that is, having more than one context entity embedded in another. These entities have values and these values are information that is available in the context

entity. An entity can have several values. The context entity is uniquely identified. This is necessary to avoid inconsistencies resulting from two context entities indicating two different context values for the same context abstraction.

The values can be associated with metadata that provide additional information about the value. These metadata are very useful when performing reasoning on the context information.

The model has four of such metadata. The timestamp identifies when context information changes. The source is the unique identifier of the entity that provides information for the values, e.g. context sensors. Probability is an indication of the trustworthiness of the information sources while user rating is an assessment indicating the values' importance to the user.

3.3.2 The Evaluator

This is the primary component of the adaptation model. Since this model is to adapt interaction between service consumers and service providers in grid environment. it is necessary to have some runtime adaptation controller that can reason about the changes occurring in the system execution context and that decide on what and how the adaptation should be executed.

Therefore, one of the responsibilities of the Evaluator is to reason on the impact of context changes on the service requests and for adapting these sets of running services by planning and selecting service variants and possibly device configurations that best suits the current context situations of the service environment [25]. The Evaluator has no prior knowledge of either adaptation strategies or context changes they affect. This means that adaptation decision must be made in a manner where users. mobile services and contexts are used to

Table 3. 1: Context Elements and their Category

ContextElement	*Context Category*	*Example Service*	*Comments*
Memory	Mobile device	Applicable to any service	To be dynamically acquired, the service is adapted to this context at the service consumer end
Storage	mobile device	Applicable to any service	To be dynamically acquired, the service is adapted to this context at the service consumer end
Service preferences	user context, application context, and session context	Applicable to any service	To be manually entered though static but temporal.
User contact/Identity Information	User Context Service Context	Applicable to every requested service	To be manually entered but static
User Location	User context	Travelling	It's either statically or dynamically acquired.
Network availability	Network context	Applicable to any service	To be dynamically acquired
Network Service Quality	Network Context	Applicable to any service	To be dynamically acquired. Can be adapted to service quality

drive adaptations. These dynamic adaptations are thus required because the mobile applications' states, device resources, service and user's requirements with their operating environment all change dynamically. The changing situations are what is regarded here as contexts and are discussed later. The context evaluator takes the context values passed to it from the Context Monitor and decides whether the values presented has changed beyond a threshold or ranges that will demand for the system to adjust itself to cope with such changes. The context evaluator uses utility function [13] for evaluation. This process and formulation are elaborated in section 3.3.2.1 below. The evaluator also takes input from the service consumer, input such as consumer preference for the requested service are used in evaluation and making feasible adaptation decisions.

3.3.2.1 The Utility Function Based Design for On-the-fly Adaptation Decision

The utility function is an objective function that expresses the values for each current state of the systems [13]. The idea of the utility function is used to model the decision making process of the system. The utility function is used as a measure of how a service or any of its variants fits a given context. It is given as a function of the properties of a service and associated context of execution. The model makes provision for expressing the adaptation policies that drive the reconfiguration module. The policies embody various parameters which change from time to time in the executing context of the system. These parameters such as consumer requirements, environmental context information and service constraints serve as input to the system.

The model considers complex relationship that may exist among various context parameters, for instance service consumer requesting for a multimedia service on her PDA will only have that service provided that bandwidth, memory and CPU processing power required by the service do not exceed what is obtainable at that point in time in the environment. The failure of any of these, even if other requirements are met, may prevent the service from being accessible. Hence, to address this ambiguity, the utility function is used to automate decision making process. This will allow automated selection of appropriate service configuration among all possible configurations of the same service so as to make the system adapt to the varying limited available resources. Since adaptation envisaged in this model is triggered by some events such as change in system resource level (CPU processing power or memory), bandwidth variation and even dynamic change in consumer requirements. three basic requirements are expected to be met.

(1) The system must know from time to time the current states of the system's execution

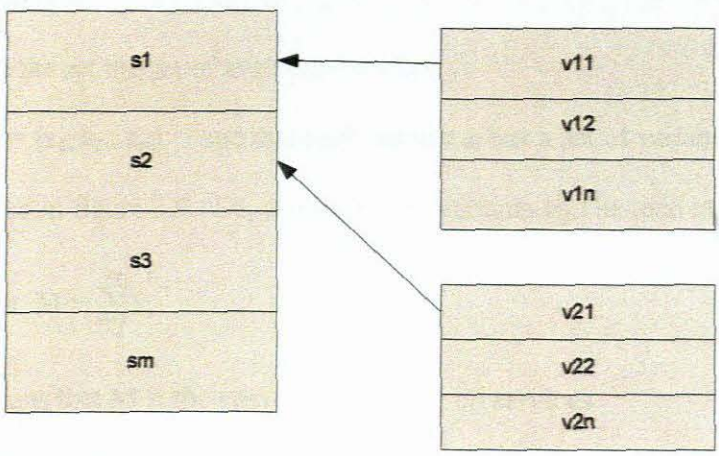

Figure 3.9: Services and Service variants Relationship

environment.

(2) The system must be able to take both proactive and reactive decisions in accordance with level of changes occurring in the system execution contexts. It may decide to reject an incoming request from a consumer or adapt to the prevailing context situation.

(3) The system must be able to provide some level of satisfaction to the service consumers as a result of service reconfiguration. These requirements qualify the utility function adoption for decision making part of this model.

3.3.2.2 Adaptation Decision Formulation

The adaptation decision is formulated using the utility function as described in section 3.2.2.1. It is assumed that a service is implemented by the service developers with alternatives that are referred to here as variants which are of different QoS requirements. Any of these variants that satisfy the prevailing context situation is selected during adaptation process. Therefore, it is assumed that CAAM processes **m** number of service requests from

service consumers. The service consumers could be a mobile application running on any mobile device.

Let **S** be the set the set of available services.

Then $\mathbf{S} = \{s_1, s_2 \ldots s_m\}$ and that each service s_i has a set of variants $\mathbf{V} = \{v_1, v_2, v_3, \ldots v_n\}$, this is illustrated in figure 3.9. So, service s_i, has variants v_{ij} The total number of service variants is given as: $M = \sum_{i=1}^{m} v_{ij}$ (4)

This means that **M** is the sum of variants for all services.

Also, let $C = \{c_1, c_2, .. c_n\}$, the set of context parameters for service execution. This will include contexts such as available device memory, processor speed or device display size that are available at any given time. And let $\mathbf{R} = \{r_1, r_2, \ldots r_n\}$, be the set of required context parameters for itch-free execution of a given service s_i then, it is assumed that each of these contexts is assigned some normalized value and that it has maximum value. Therefore, each service s_i will require r_{ij} contexts for execution. Let x_i be the decision variable that indicates if a particular variant v_{ij} of service s_i is selected for adaptation.

Therefore, the sum of available contexts c_{ij} must be greater than or equal to the sum of required contexts r_{il} of service variant s_i. The runtime utility for each service variants v_{ij} is calculated by: $u_{ij} = r_{ij}/c_{ij}$ and the utility for each service s_i is thus calculated as

$$U_i = \sum_{i=1}^{n} r_{ij} / c_{ij} \quad (5)$$

In order to ascertain which of the variants of service s_i is selected, let d_{ij} be the variable that ascertains that service variant v_{ij} is selected and x_{ij} be the decision variable which is set to 0 or 1 depending on whether v_{ij} is selected. Then, we have $\sum_{i,j=1}^{n} d_{ij} x_{ij} = 1, \ \forall i \ \forall j \ x_{ij} \in \{0,1\}$

. (6)

That is, for any given service s_i, only one of its variants v_{ij} would be selected for adaptation. Each service variant must satisfy some quality or preference requirement, indicated by attached weights w_{ij}. The sum of w_{ij} for a given service s_i must be 1:

$$\sum_{i,j=1}^{n} w_{ij} = 1 \quad (7)$$

Finally, we have the following formulation for decision making model. We need to maximize

$$U_{(uij, xij, wij)} = \sum_{i=1}^{N} \sum_{j=1}^{N} u_{ij}(w_{ij}) x_{ij} \quad (8)$$

Subject to the following constraints:

$$\sum_{i,j=1}^{N} u_{ij} \leq u_i \text{ for } n = 1,\ldots, N \; \forall_{ij} \in \{0,1\} \quad (9)$$

Where u_{ij} is the utility parameter associated with each service s_i and N represents the number of utilities under consideration.

$$\sum_{i,j=1}^{N} d_{ij} x_{ij} = 1, \quad (10)$$

This means that we need to select a service variant v_{ij} of variants for service s_i with the highest utility that satisfies the constraints above. The utility function formulation is an extension of the multidimensional Knapsack problem [22] with choice problem constraints.

As formulated in the utility function, the decision on which appropriate service variant to execute is based on varying context values at any given time. Two algorithms could be used to solve this problem according to Khan in [36]. One, the exact algorithm, as a result of its high computational complexity, cannot be applied particularly for runtime adaptation

decision making being considered. Therefore we need to consider heuristic algorithms which can help arrive at decision within reasonable computation time with feasible and optimal decisions.

3.3.2.3 The Decision Making Algorithm

The decision making problem formulated above does not have an exact solution since it's a variant of the popular Multidimensional Multi choice Knapsack problems [34]. There exist various heuristics approaches to solve this type of combinatorial optimization problem. The aim of the above formulation is to pick from among service variants v_{ij} that satisfies the context constraints. A heuristic algorithm based on the work of Michrafy and Sbihi [35] that provides a guided local search heuristic algorithm GLS that has its origin from constraints satisfiability applications is adopted. Proved that this algorithm is very effective at solving this type of problems. The algorithm uses memory to guide the search for appropriate service variant. The algorithm comes up with a feasible selection of service variant v_{ij} such that $\sum_{i,j=1}^{N} u_{ij}(w_{ij})x_i \leq u_i \quad \forall ij,\ x_{ij} \in \{0,1\}$ *for* $i,j = 1 \ldots N$ and for each service s_i with variants v_{ij}, we pick service v_{ij} such that $\sum_{i,j=1}^{N} d_{ij} x_{ij} = 1$ and 0 otherwise.

For m = 1 to M, set $u_i = \max\{u_{ij}, i,j = 1 \text{ to } N\}$
Si ← vij;
set s(m) = s_{ij}; $d_{ij}x_{ij} = 1$
set $C_i = \sum_{i,j=1}^{N} u_{i,j}x_{ij}$,I,j=1,...,M;
end For;
s = (s_1,...,s_n)
while ($u_{ij} > u_i$, for j = 1 to N){
n_0 ← argmin {u_{ij}} 1≤i≤n
m_0 ← argmin{u_{ij}} 1≤m≤n
y[i0] = vi_0; $d_{ij}x_{ij} = 0$;
$u_{ij} = u_{ij} - u_{i0}$ for m = 1 to M;
if ∃ $v_{i0} \neq vi_1$ and $\sum u_{i,j} < u_i$, for i,j = 1 to N
then
$d_{ij}x_{ij} = 1$;
$v_{i0} = v_{i1}$

v_{ij} = (v_{i0}; si, $\forall i \neq i_0$, i =1 to M) is feasible
return v_{ij};
endIf
EndFor;

S'm0 ← argmin{u_{ij}};
vi_0 = s'; s_{i0} = si; $d_{ij}x_{ij} = 1$;
endwhile;
return S.

Figure 3.10: The adaptation Decision Algorithm

The feasible solution is obtained by a modified greedy procedure of [34]. The modification done is that we do not need to add any penalty for choosing an infeasible solution in the first place. This is not necessary in our context as the algorithm selects a feasible solution from the calculated utilities for each service variant. This it does by evaluating the ratio: $\mathbf{u_{ij} = r_{ij}/c_{ij}}$ where i,j,l∈ {1,....N} of each context parameter c_{jl} required by each service variant v_{ij} and (c_{ij},w_{ij}) is the scalar product of both c_{ij},. The algorithm then selects service variant v_{ij} for service s_i such that vij∈ {v_{11},v_{12},....,v_{nm}} with lowest u_{ij} is the selected service variant that is

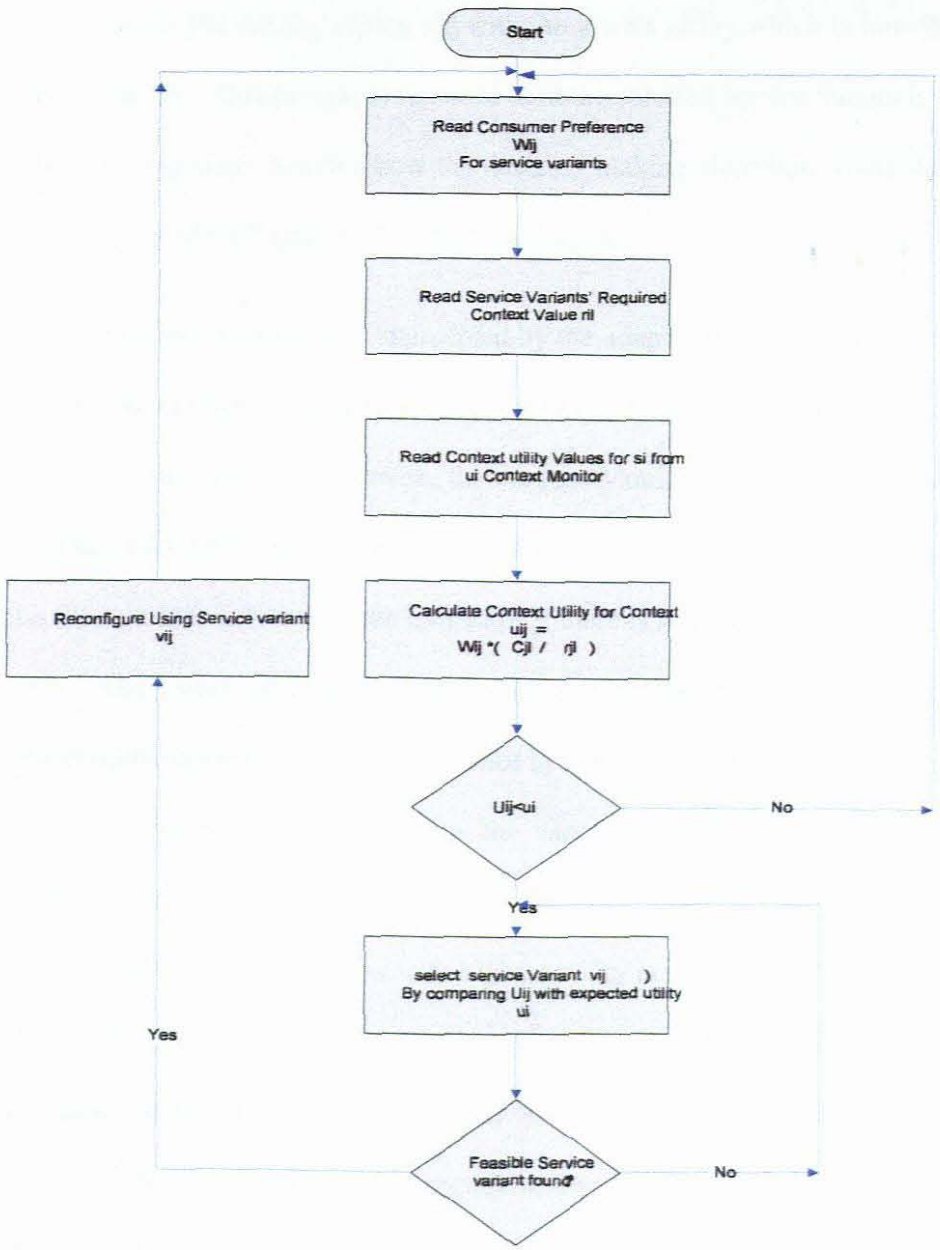

Figure 3.11: Adaptation Decision Flow Diagram

feasible under the current context situation then the algorithm terminates. Otherwise, it takes context c_{j0} as not being satisfied. It then with respect to c_{j0}, selects service variant v_{ij} corresponding to s_i with lowest utility u_{ij}. This very service variant v_{i0} is then swapped with

another service variant say v_{i1} and so on. If the new selected service variant is not feasible then it selects the service variant v'ij with the lowest utility which is now the newly selected service variant. This process is repeated until the selected service variant is feasible.

The following steps describe how the decision making algorithm works which is also given in Figure 3.9 while Figure 3.11 is the flow diagram.

(a) The service request is intercepted by the adaptation service that subsequently extracts relevant user context information and passes these to the context Monitor.

(b) In the initial configuration, the context Monitor sends necessary context values, for instance, user preference information to the Context Evaluator. In reconfiguration situation, the Context Monitor notifies the Evaluator if there is any significant change.

(c) The Utility function for each of the service variants is calculated using the required and available context with user preference in terms of weight.

(d) Having calculated the utility, the variants are arranged (sorted) according to each context utility.

(e) Then the service variant with highest utility is selected.

(f) As the service variant is selected, other incoming service requests must also be processed which may have some impact on the adaptation service agility [33]. For example, if the service variant selected will require more of that particular context to execute, and then this will leave other requests with less of it to execute. This may trigger recalculation of the utility function for the unselected service variant.

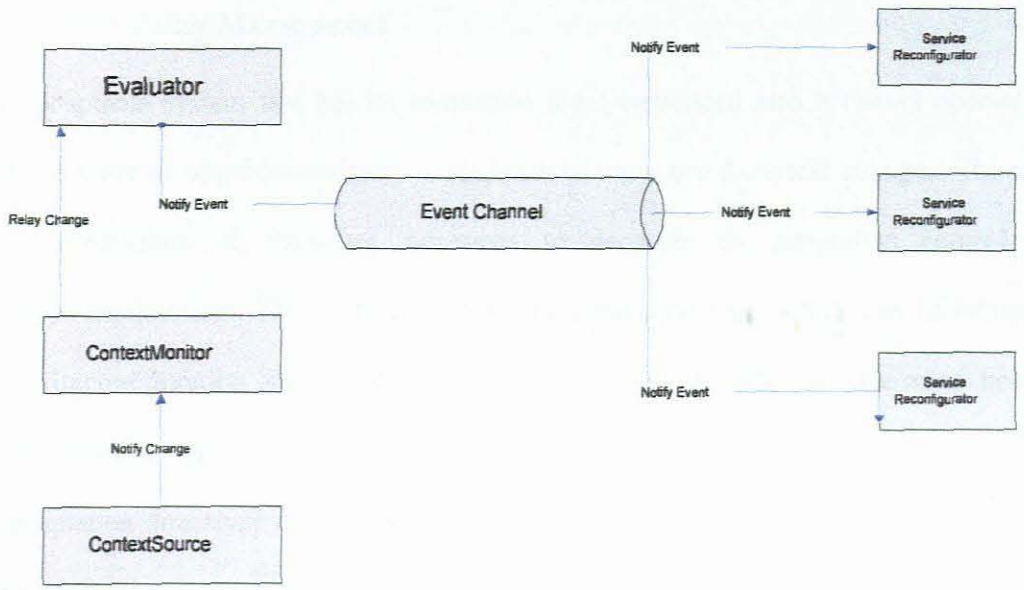

Figure 3.12: Communication Model between Components of the Model

This could be avoided by choosing the variant for the next request if there is no significant secondary effect for choosing such variant.

(g) However, if the adaptation service recalculates the utility for the new request and sorting its variants, the adaptation service selects the next variant with the highest utility; the previously selected variant is not reconsidered in the new calculation. This algorithm helps determine which of the variants of a requested service best suits the present prevailing context situation.

The decision reached by the algorithm is passed to the policy Manager (see figure 3.6) that determines the appropriate adjustment to be effected on the system through the reconfiguration module. The policy manager defines a mapping of the results of the Evaluator of the monitored context and resource parameters to components configuration.

3.3.2.3 Utility Policy Management

Every adaptable system that has its adaptation logic embedded into it cannot operate in a generic manner or adapt dynamically in response to unexpected context changes. The use of policy management is, therefore, necessary to decouple the adaptation control from adaptation mechanisms. This helps to specify the adaptation logic which can be interpreted by the Reconfiguration module dynamically at runtime in order to determine how the systems should adapt.

The adaptation directives need to contain some necessary elements such as what to adapt, which adaptation to apply, how and when this should be applied, and what constraints may limit the application of such adaptation. The use of these directives or adaptation policies will allow a declarative specification of how the system would adapt without necessarily specifying how this adaptation would be accomplished. This allows decoupling of adaptation logic and the adaptation mechanisms. The policies are formulated by using Event-Condition-Action (ECA) format [27] and these policy utilities are in two forms; the reactive and proactive forms. The reactive format is specified by selecting an event to be fired in response to context change, adaptation target and adaptation to be effected on the target. The application of the utility-based policies needs to be provided with some set of statements that needs to be evaluated when the policy rule is fired. On the other hand, the proactive format specifies the adaptation target, adaptation to apply and a set of conditions that are triggered immediately upon observing some pattern of change in contexts. The utility-based policy [28, 74] is adopted as it allows on-the-fly adaptation decision and it does not create much overheads since decision are not predetermined but are made dynamically based on the utility of the current context situations. The Utility policy management is responsible for the

interpretation of adaptation decision, conditions and the activation of appropriate adaptation.

3.3.3 Designing the Service Component and Reconfiguration model

Grid service must grow to meet increasing requests, new requirements, and new applications. Flexibility and availability usually conflict with each other, however, service providers in conventional systems do shut down in order to reconfigure and then restart the service. In the utility computing paradigm, it is unacceptable to disrupt the service execution even for two seconds. Dynamic reconfiguration, identified by Otebolaku in [79, 81], therefore, proffers solution to managing runtime changes of service execution without shutting down or disrupting the service execution. In this section how the proposed model integrates component based technology and dynamic reconfiguration for effecting adaptation is discussed.

3.3.3.1 The Component Model

The applications are assumed to be component-based. They use and provide resources and services as they operate in contexts. These applications are influenced by the changes in resources they require or provide and by the context of their execution. Therefore, we adopt component based design [11] as it offers a standard mechanism for carrying out reconfiguration functions such as addition, removal and deletion of components that are affected by resource variability and context changes. This component model is illustrated in Figure 3.13.

Each of the components has two types of interfaces: *provided* and *required*. Two or more components interact via connector that is hooked to the components through its interface. A service component may provide one or more services to the mobile Grid client's component

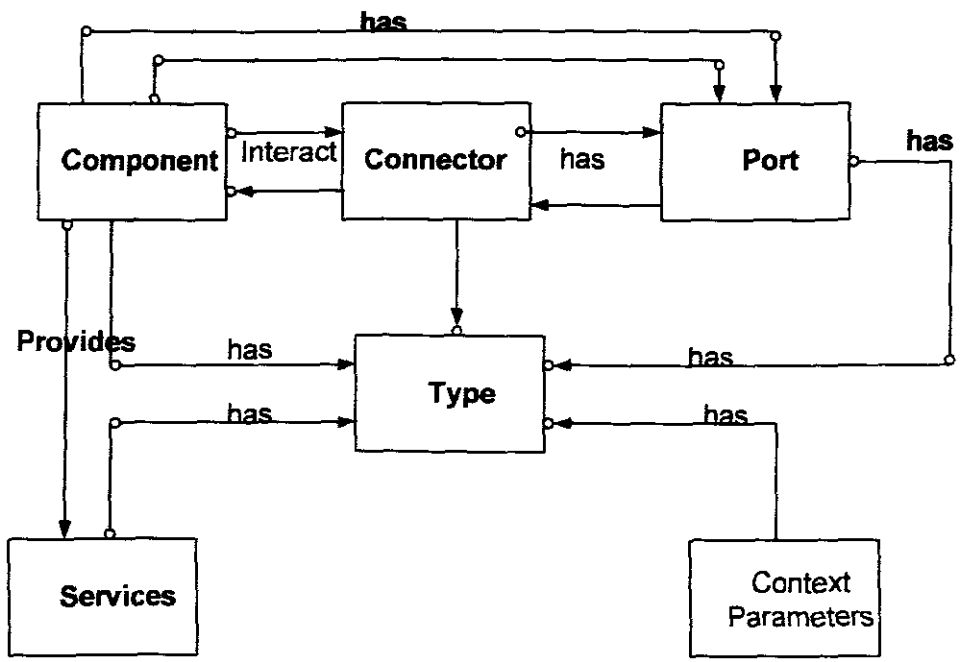

Figure 3. 13: CAAM Component Model

provided that context and resource variability do not exceed certain required level. One of the main benefits of using components frameworks is that it helps to regulate the component roles and standardizes interfaces. The component model used is based on the one designed and used in the **MADAM** project [25].

A ***service component*** is designed as a unit of composition with contractually specified interfaces and explicit dependencies where dependencies are specified by stating the required interfaces and the acceptable execution platform. Each of these components can be atomic or composite. In this way, a service can be assembled from a recursive structure of component frameworks as illustrated in figure 3.12. Services are therefore defined as interfaces that allow specifications of behavioural features. In this component model, interfaces are organised as ports and port types. A ***port*** represents a component's capability of participating

in a specific interaction. Distinct components can define ports of the same types meaning that they provide or require identical interfaces. The port ***type*** characterises the set of services that are provided or required through ports of that type. The component framework describes collaboration of components by defining roles for each component participating in the collaboration and their interaction. Each role is described by a ***component type*** which defines the functional behaviour and attributes of the **component**. The component type defines a set of ports that a component of that type must implement. Each of the components has two types of interfaces: provided and required. Two or more components interact via connector that is hooked to the components through its interface.

In order to bind the components together, connectors are used. These ***connectors*** specify the interaction between a required port of a component and a provided port of another component. The former is used for request while the latter is used for responding to a request. The interfaces defined by the two ports define the syntax and semantics of interaction between the components over that given connector. An instance of these connectors is the implementation of a connector enabling interaction between two component instances. When components are instantiated in the same address space, then the connector is a mere local object reference such as java object reference. Otherwise, a connector is realised by remote communication protocol such as SOAP, Java RMI, RPC, etc. The connector instances are created by the connector factory [25]. Creating connector instances means providing each component that specifies a requirement for an interface provided by the other component.

3.3.3.2 The Run-Time Component Model

The run-time model of MADAM [25] is adopted and used to describe different aspects of the service entities. This run-time model is what is used by the adaptation model to reconfigure

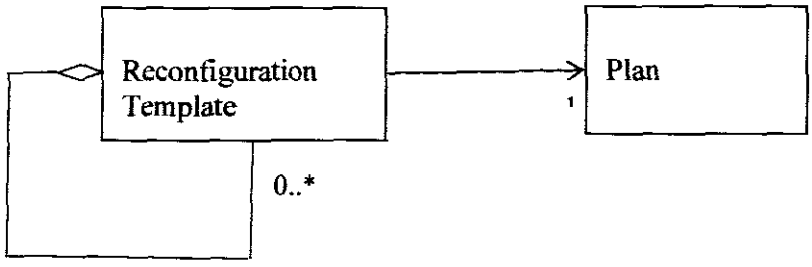

Figure 3. 14: Reconfiguration Template and Plan Relationship

the running service component at execution time. Each component is presented at runtime as ***component plans*** that specify the behaviours of a component and the required implementation resources for that component. Therefore, there exists a relationship between component, port, plans and connector types. Several service components can implement same component type but the specifications of differences in the offered and needed interfaces must be ascertained. All that service developer does is to publish several plans that are associated with a component type, where the plans specify the properties. We adopted the plans available in MADAM Middleware. These plans are discussed in the subsections that follow.

3.3.3.2.1 Blueprint Plan

This is serialised, immutable implementation artefact that is used to load components at runtime. This plan specifies the resource requirements. For example, Java class requires JVM to execute, thus plan must specify a resource requirement to a Java-enabled run-time environment. Also, this is where resource requirements of service variants are kept for calculating service utility.

3.3.3.2.2 The Instance Plan

The instance plan is associated with the existing component instance. These are used when a component instance can be shared or reused.

3.3.3.2.3 The Connector Plan

This describes the implementation of a connector. A connector plan implements a connector type, and can be used to instantiate this type between components.

3.3.3.2.4 The Composition Plan

This describes the components implemented as collaboration between several components. It defines the roles of components participating in such collaboration. For instance, in a multimedia service application, this could be implemented as a composition of a multimodal Interface components, multimedia storage and persistent data retrieval components.

3.3.3.3 The Reconfiguration Template

When changes occur in the context of execution of a given service, some inconsistencies may follow, for instance, if a running service goes to offline mode, it may decide to switch off the network adapter in order to save battery. If another executing service is using this network adapter, then there is conflict. In order to resolve this conflict, reconfiguration template is used to ensure consistent state of the context aware system. It provides the intentional representation of the set of service variants, and that of the model of these service variants where all variation points [25] are resolved. So, the reconfiguration template is needed to provide representation and can also be used to evaluate the variants and to instantiate them. The configuration template refers to the plan for which it provides additional information to resolve the variation point. In its simplest form, it refers to the blueprint plan without

parameters settings. When it refers to composition plan, it will contain mapping from a role in the composition plan to a nested configuration template that resolves the variation point of the given role. A role describes the functionality of a component. It could also refer to blueprint plan with parameters, in which case, it contains reference to a selected parameter and device setting and corresponding properties. Each of these parameters is considered as a different realisation of the components and therefore corresponds to different configuration templates.

Whenever a reconfiguration template representing different service variants is being created, the architectural constraints applied to the composition plan are checked. If the constraints are not satisfied, the configuration template will be discarded and will not be used for adaptation purpose.

3.3.3.4 The Service Reconfigurator

The *Reconfigurator* is responsible for coordinating the initial configuration and reconfiguration of the service components and the device. The Evaluator and the *Reconfigurator* work together as they operate on some common information element, which is the configuration template discussed in section 3.3.3. When adaptation of a service is being carried out, the *Reconfigurator* proceeds according to the configuration template for the variant selected by the Evaluator. Thus, the *Reconfigurator* carries out the adaptation decided by the Evaluator by applying the configuration template. When, carrying out configuration and reconfiguration, the *Reconfigurator* uses the components frameworks interfaces to instantiate components and connectors for a given service.

To achieve this, service reconfiguration pattern [77, 79] is adopted because it decouples the service functional behaviour from the point in time at which these service implementations

are reconfigured. This decoupling improves modularity of services to evolve over time independently of reconfiguration issues such as whether two services must be collocated or what concurrency model should it use to execute services. Also, service reconfiguration pattern centralises the administration of services it configures. This facilitates automatic initialisation and termination of services and can improve performance by factoring common services initialisation and termination into efficient reusable components.

Further, the pattern is useful when a service needs to be initiated, suspended, resumed, and terminated dynamically. It is also used when service configuration decision are made at service runtime especially for dynamic adaptation of services. It is a very efficient, flexible and convenient way of implementing distributed services. Service reconfiguration pattern decouples service behaviours such as their interaction from the point in time at which they are configured into application.

In considering service reconfiguration, here are some of the problems it helps to solve in this model:

(i) When there is need to defer the selection of a particular implementation of a service till runtime. This will allow service developers to concentrate on service functionalities, without consuming much time in committing themselves to any particular service configuration. It in essence decouples non functional behaviours of a service from functional behaviours which makes the service to evolve independently of the reconfiguration policies and mechanisms used by the systems.

(ii) The need to build complete applications or systems by composing multiple independently developed services that do not require global knowledge. The service reconfiguration pattern requires all services to have uniform interface for configuration and control. This

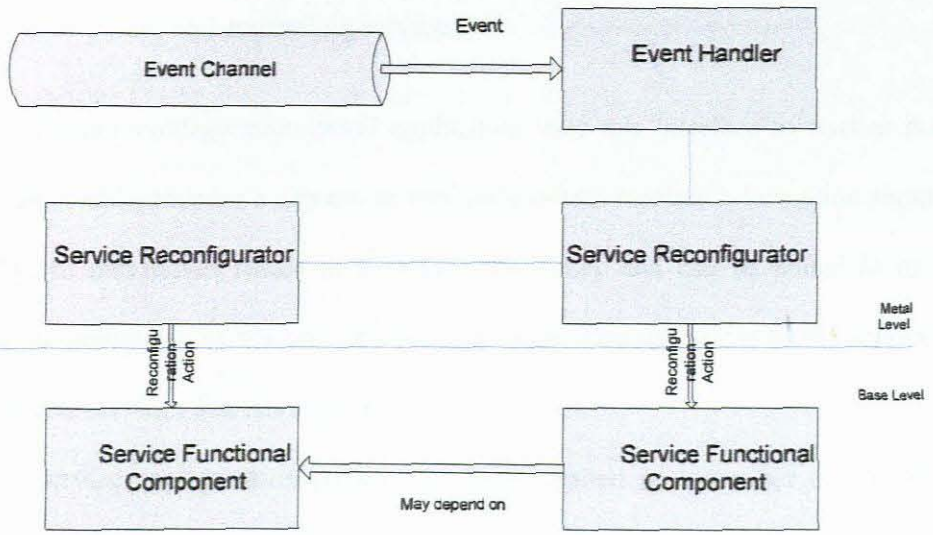

Figure 3.15:Reconfigurator Model

allows the services to be treated as building blocks that can be integrated easily as components into a larger application. The uniform interface across all services makes them look the same with respect to how they are configured.

(iii) This uniformity in turn simplifies the distributed application development.

(iv) The needs to optimise, control, and reconfigure the behaviour of a service at runtime. Decoupling the implementation of a service from its configuration makes it possible to fine-tune certain implementation or configuration parameters. For instance, depending on the parallelism available on the hardware and operating system, it maybe either more or less efficient to run multiple services in separate threads or processes. The service configuration pattern enables applications to select or tune these behaviours at runtime, when more information maybe available to help optimise the services. In addition, adding a new or updated service to a distributed system can be performed without requiring downtime for existing services. Figure 3.15 illustrates the relationship between service functional components and the *Reconfigurator* that provide standard interface for

configuring and controlling services.

A service reconfiguration based application uses this interface to start or initiate, suspend, resume, and terminate a service, as well as to obtain runtime information about a service. The services themselves reside in a service repository and can be added to or removed from service repository by the service *Reconfigurator*. The subclasses of the service base represent concrete services that have specific functionalities.

The service configuration pattern has been applied in a number of scenarios particularly when:

(i) Services must be initiated, suspended, resumed, and terminated dynamically at runtime,

(ii) The implementation of a service may change but its configuration with respect to related services remains the same and the configuration of collocated services may change but their implementations remain the same,

(iii) An application or system can be simplified by being composed of multiple independently developed and dynamically configurable services and

(iv) The management of multiple services can be simplified or optimised or adapted by configuring these services using some administrative unit.

The key components of this pattern are:

(a) Service: This specifies the interface containing the hook methods such as initialisation, remove, suspend, terminate used by the service to dynamically configure the service at runtime,

(b) Concrete Service: The concrete service implements the services interface and functionalities such as event processing and communication with service consumers.

(c) *Service Repository*: This maintains a repository of all services offered by a service

Reconfigurator based applications. This allows administrative entities to centrally manage and control the behaviour of the configured services. The service configuration initialises a service component by calling its *init()* method. Once the service has been initialised successfully, the Service *Reconfigurator* adds it to the service repository that manages and controls all the services. After the service is configured into an application, a service performs its processing tasks i.e. servicing service consumer's requests. Then the services *Reconfigurator* terminates the service once it is no longer needed by calling *finit()* method on the service. This hook allows the service to clean up before terminating. As soon as the service is terminated, the service *Reconfigurator* removes it from the service repository.

3.3.4 Adapting Services

The sequence of steps followed to adapt services is explained based on the previously discussed designs. These steps are as follows are as illustrated in Figure 3.16.

(a) *getContext():* In order to evaluate service variants, the Evaluator requires the current values of some of the relevant context values for the given service.

(b) *conextValueChanged():* The Monitor informs the Evaluator of a change to a context element for which it has previously requested a change notification.

(c) *configure():* Having decided which variant suits the current contexts, the Evaluator requests the *Reconfigurator* to perform reconfiguration.

(d) *disconnect():* While reconfiguration is ongoing, the *Reconfigurator* will call this operation of the component framework to remove existing connections between component instances, which will not be part of the configuration for the new set of concurrent services.

(e) *init():* The *Reconfigurator* calls this operation to create all new components instances

required in the configuration for the new set of concurrent services.

(f) *resolve()*: The *Reconfigurator* requests the components framework to resolve the name to an initialised component, which may cause a proxy to be created in case the component is remotely located.

(g) *connect()*: Each new connection between components of the new configuration of the concurrent service variants will be created by calling this operation, and for this to work, the *connectorFactory* to use is specified.

(h) *ReconfigureService()*: The Evaluator instructs the *Reconfigurator* to reconfigure the service to the new set of utility values for the service context parameters.

Finally, the next section provides insights on how the design explained in this chapter achieved some of the set goal and objectives of this investigation.

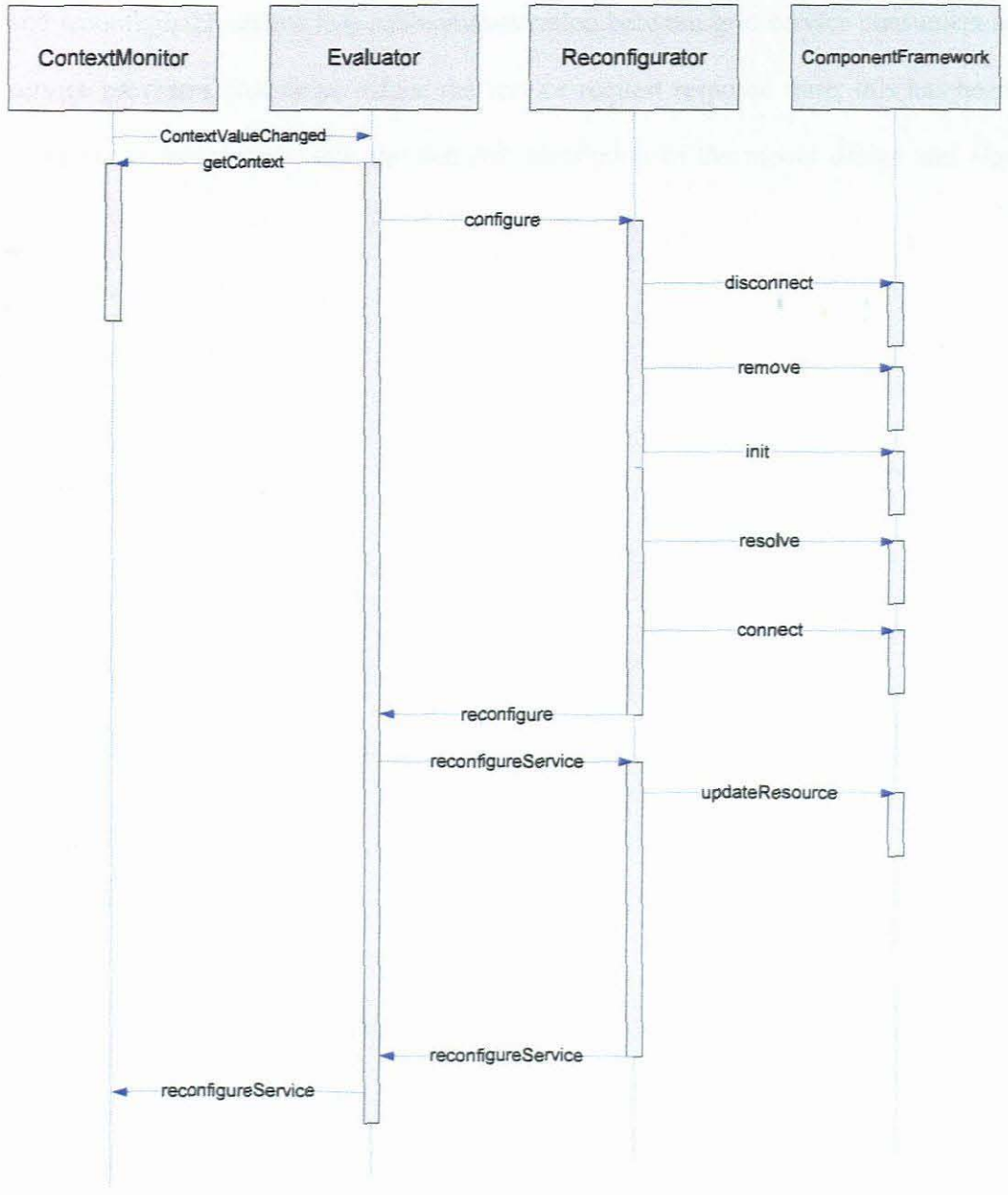

Figure 3. 16: Adaptation using Reconfiguration Pattern

3.3.5 Chapter Conclusion

In this chapter, we have described the design of a dynamic system for service interaction in a mobile grid system. As stated in section 1.4, the goal of this work is to propose an adaptable

and reconfigurable interaction and communication between grid service consumers and grid service providers, that helps reduce the service request response time; this has been partly achieved in this chapter with the detailed description of the model design and algorithm.

CHAPTER FOUR

MODEL IMPLEMENTATION AND EVALUATION

4.1 Introduction

This chapter describes the implementation of the idea and adaptation model design presented in previous chapters as a proof-of-concept. Most part of this chapter describes how the *Context Monitor*, *Evaluator* and the *Reconfigurator* are realised. The chapter ends with the evaluation of the simulated service that runs in the developed CAAM environment.

As explained in chapter three, the main goal of the adaptation model is to propose a dynamic and adaptable system for service interaction in mobile grid. The objectives to achieve this were revisited in chapter three in order to ascertain how these were achieved in the model design. The model consists of a context monitor that further contains context sensors, and a persistent context storage that temporarily stores context values that are not consumed immediately for determination of trends of context changes, and also for aggregation of further context data from the original contexts. The *Evaluator* uses '*eventing*' to communicate its decision to the *Reconfigurator*. The MADAM (Mobility and Adaptation enAbling Middleware) [25] provided us with the environment, APIs and libraries to implement major parts of the model.

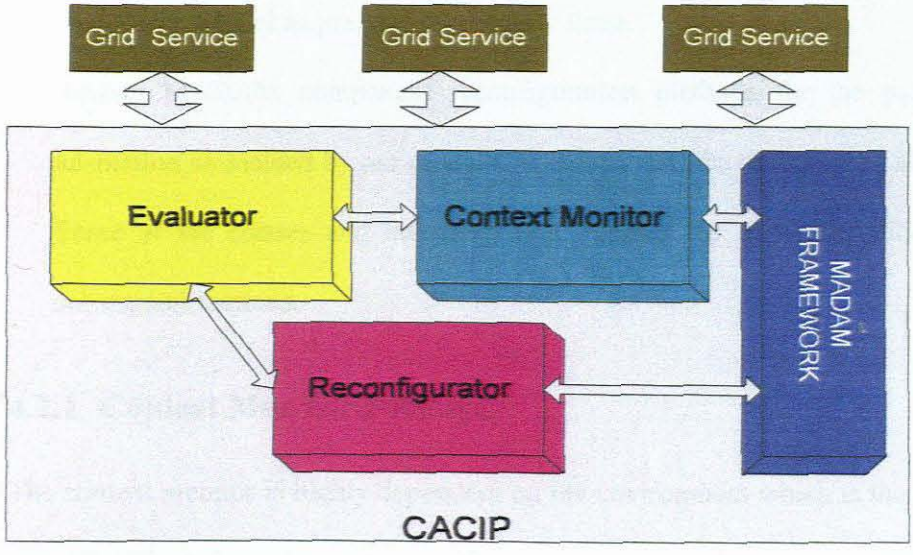

Figure 4.1:The Implementation Framework

4.2 Implementation Framework

The design of the components of the adaptation framework has been discussed in chapter three. However, the implementation of these components integrates the MADAM APIs and libraries that provide useful implementation of some of these concepts. Figure 4.1 illustrates the framework and how it integrates with the MADAM to achieve our set Goal and objectives. Next, the integration of the adaptation model and MADAM is briefly explained thus:

1. The context Monitor communicates with the MADAM *ContextSensor* interface in order to retrieve actual context information from MADAM Context and Resource Managers that generates these from low level device resources. This helps reduce the implementation time and to avoid reinventing the wheel.
2. Making use of the MADAM component metal-model which helps to integrate MADAM component types, connection plans, blueprints plans, and composition plans into our

component Model as presented in chapter three.

3. Adding MADAM component reconfiguration methods for the purpose of effecting adaptation as decided by our model's *Evaluator* and the utility based algorithm.

   Some of the classes and interfaces that realised the implementation are presented in subsequent sections.

4.2.1 Context Monitor Package

The context monitor is highly dependent on the environment which is the physical context of the terminals and wireless network. Therefore, the dependencies are encapsulated in context sensors from which context data as observed in the environment is generated. Some of these context data is consumed directly as it is generated by the Context monitor. It then passes this to the *Context Evaluator*. Those context data that are not utilized immediately are stored temporarily in the persistent context store and are retrieved later by the Evaluator through *contextAccessInterface* interface and used for proactive adaptation decision.

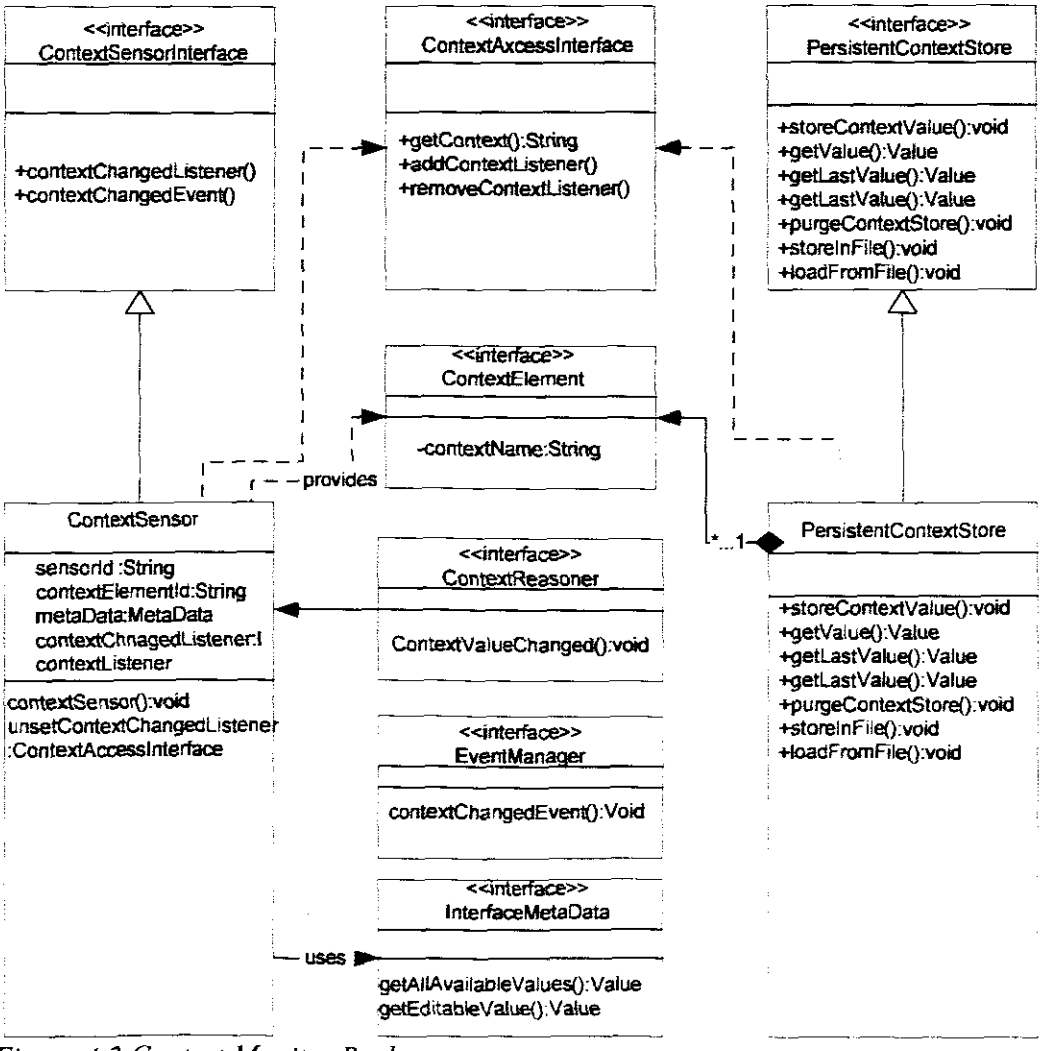

Figure 4.2:Context Monitor Package

The context Monitor package as specified in chapter three will be responsible for monitoring the context and resources in the service execution contexts using *ContextSensor*, *PersistentContextStore*, *ContextAccessInterface*, *contextSensorInterface*, *eventListener* interface, *contextElement* class, the context metaDataInterface and the *persistentContextInterface*. These interfaces and classes work together to provide context monitoring functionalities and Figure 4.2 is the class diagram for the context monitor.

4.2.1.1 ContextSensor Class

This is responsible for sensing contexts. The *ContextSensor* class provides context values to the Evaluator package and also to the persistent context store. This *ContextSensor* class can provide more than one context element. The Evaluator acts as clients to the *ContextMonitor* package through the *ContextAccessInterface*. This class also implements the *ContextSensorInterface*.

4.2.1.2 ContextSensorAccessInterface

The *ContextSensorInterface* is implemented by the *ContextSensor* class. It contains context sensorID field which identifies each context sensor in case there are more than one. *contextElement* field is the context element(s) being monitored by the context sensor. The *contextValueChangedEvent()* is fired whenever value of the context element being monitored by the context sensor changes. *ContextChangeListener* is the interface on which *contextValueChanged()* listens for context changes.

4.2.2 Evaluator

Section 3.3.2 of chapter three presented the design and function of the Evaluator as being responsible for reasoning on the impact of context changes on the services, and for deciding on the adaptation to effect if context evaluation necessitates adaptation. This it does by planning and selecting the service variant that best fits the current context situation. Part of the reasoning requires that the Evaluator accesses the utilities of the contexts of the service variants. The Evaluator then produces a model of the service variant that best fits the current contexts using the decision algorithm. It then uses the configuration templates to resolve all the variation points. The Evaluator uses the context Monitor interfaces to collect context

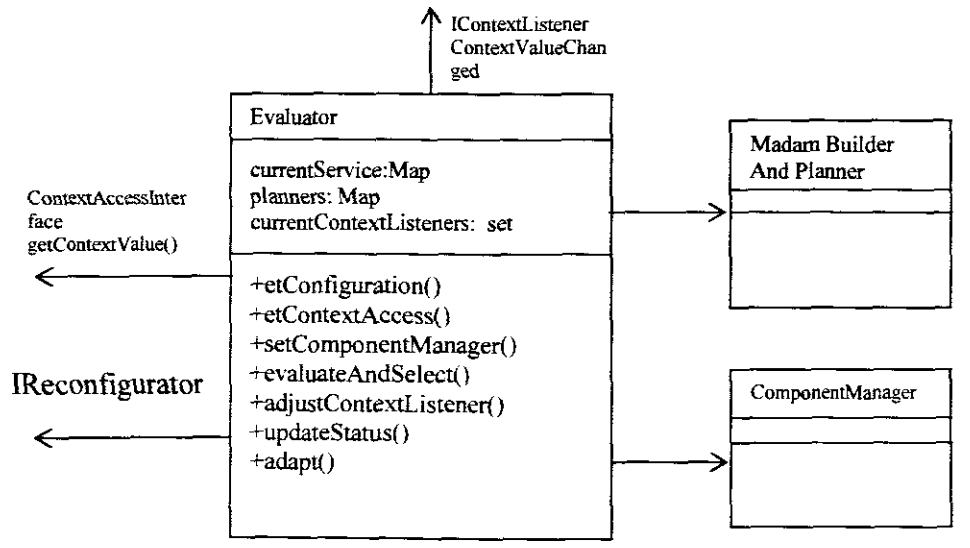

Figure 4.3: Evaluator Class Diagram.

information on the networks and devices. The Evaluator class implements the *ContextAccessInterface* to retrieve context information it requires from the persistent context store or directly from *ContextMonitor* package. The class diagram is given in Figure 4.3, it illustrates classes and methods and interfaces that implement the Evaluator. More importantly, the *Evaluator* implements the *ContextListenerInterface* so that it reacts at runtime to context information for currently running services. It also implements the *ComponentManagerInterface* to acquire relevant plans that constitute the service architectural model. The main event that triggers the *Evaluator* is *contextValueChangedEvent*.

The Builder and Planner is the MADAM [25] API used to build the adaptation framework or request necessary adaptation decision from Evaluator. The ***adapt()*** method triggers the adaptation by making the reconfigurator to initiate the process of adaptation. This is

accomplished by first acquiring current context from the context Monitor using the *ContextAccessInterface*. Then, *evaluateAndSelect()* method is called to iterate over all possible configuration templates for the running services using *buildTemplate()* method of MADAM *BuilderAndPlanner* API. Only combinations of service variants that fit best in the current context and resource limits are considered and the best among these is selected by calculating the combined utility of each of these combinations. When the choice has been made, the *Reconfigurator* is triggered to adapt the running service and its selected variant is composed and presented to the service consumer.

4.2.3 The Service Reconfigurator

The service *Reconfigurator* coordinates the initial configuration and the reconfiguration of the service components. The *Evaluator* and the *Reconfigurator* operate on common information which is the configuration template. In the course of adaptation, the *Reconfigurator* normally proceeds according to the contents of the configuration template for the selected service variant. Therefore, the *Reconfigurator* executes the adaptation that has been decided upon by the Evaluator while it applies the configuration template. Furthermore, the *Reconfigurator* uses the component framework interfaces to instantiate components and connectors to disconnect from any other component.

The *ServiceReconfigurationInterface* provides the operation to request the configuration of a service. A configuration template that models the variant to be instantiated is provided to the *Reconfigurator*. The *Reconfigurator* uses the connector Interface to create and remove connectors using its ***addConnection()*** and ***removeConnection()*** methods. The new configuration of the service is provided to the *ServiceReconfigurator* has a set of

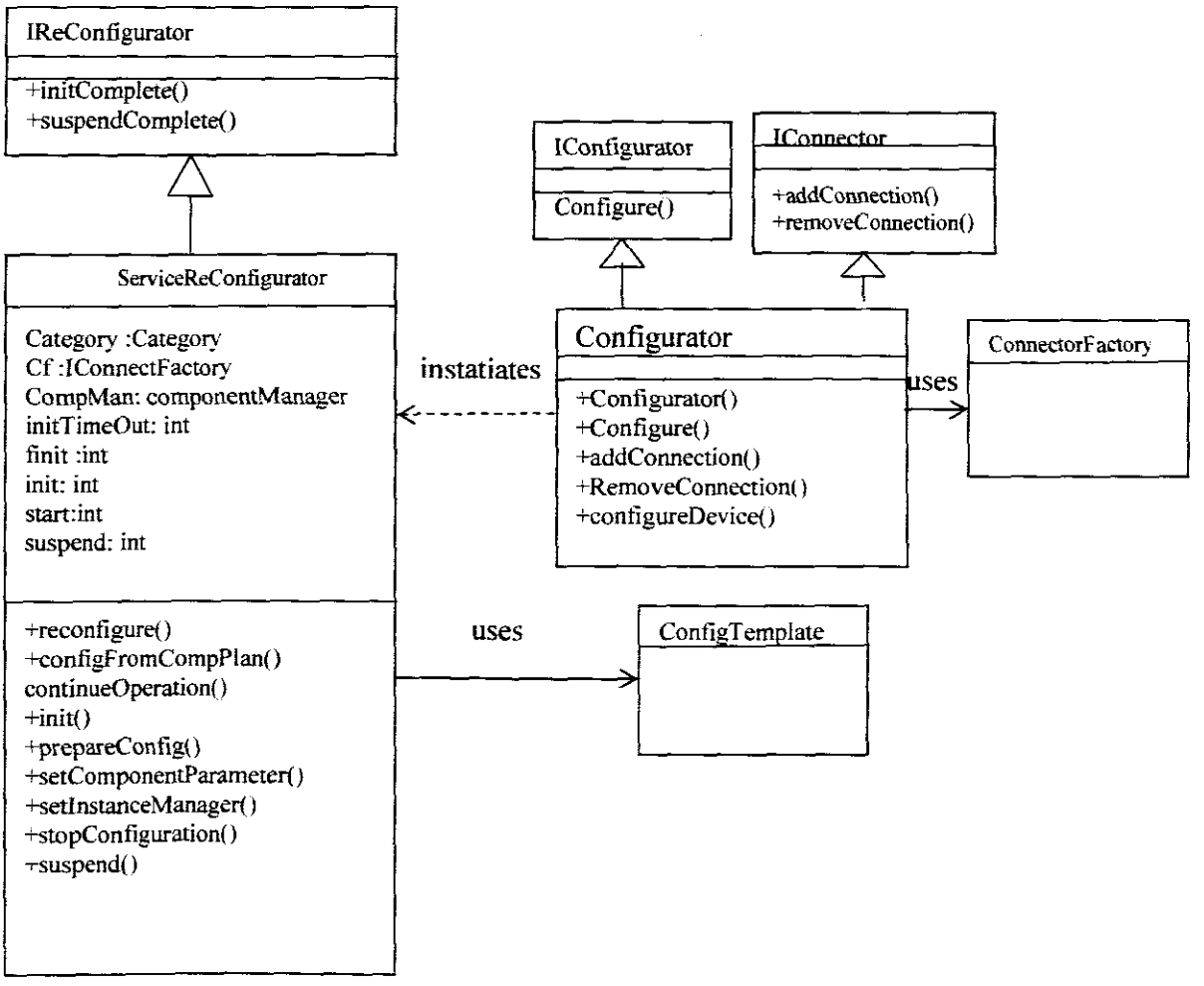

Figure 4.4: Reconfigurator Class Diagram

configuration templates, one for each running service. The components for each service and its structured subcomponents, their blueprints and connectors used for each component, and parameter setting for each component are included in the configuration templates. The service *Reconfigurator* maintains information of the currently running service variants. Whenever a request for service reconfiguration is received, the new configuration templates are compared with the information of the currently running services to derive the minimal sequence of steps required to move from the current reconfiguration of the service to newly

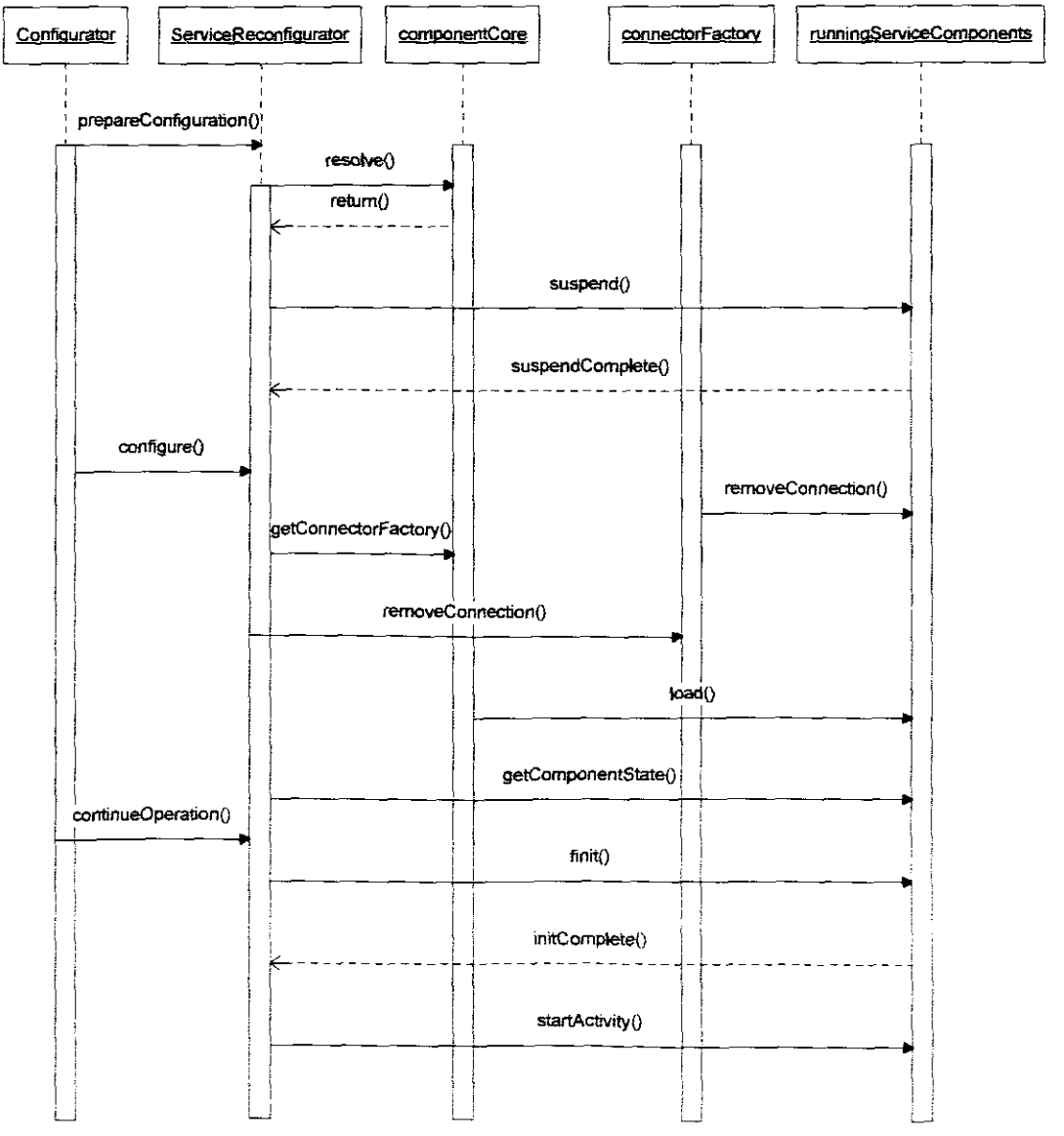

Figure 4.5:Sequence Diagram for Reconfiguration Process Summary

described Configuration by the reconfiguration templates. When connecting components, the service *Reconfigurator* uses information from the reconfiguration template to determine the connector to use. The reconfiguration template contains the model's component variants which can be used to instantiate components. The service *Reconfigurator* also determines

whether to use existing connectors or to create new ones using the connector factory [25]. Figure 4.4 is the class diagram for the *Reconfigurator* illustrating its interfaces and classes. The *Configurator class* is responsible for the initialisation of a service. The *Reconfigurator* class receives configuration templates from Evaluator that contains all necessary information needed to execute a proper reconfiguration. Its *configure()* method uses the reconfiguration template that is provided to the *serviceReonfigurator* class when it is instantiated by the *Configurator* class in order to reconfigure the service.

prepareConfig() method prepares the running service for reconfiguration by suspending all component of such service while ***configFromCompPlan()*** reconfigures the service from the composition plan. ***stopConfiguration()*** is invoked to the current component variant of currently running service. The serviceReconfigurator class keeps the states of the service being reconfigured using the ***init, finit, start, suspend*** parameters. These are to store the states of the service components when being started and suspended. Figure 4.5 summarised the reconfiguration process.

4.3 Implementation Environment and Specifications

To realise implementation of the prototype, we made use of MADAM reconfiguration framework, Resource and Context Management APIs as explained in section 4.1. In addition, the prototype is implemented with Java programming language using the Netbeans Integrated Development Environment version 5.5.1. The persistent context store was implemented as a simple text file where context values can be written and read. This is because we intended to keep this implementation as simple as possible and also to reduce the overheads of using desktop database which is an implementation is carried out on a Pentium 4 Compaq Presario

The context elements described above will trigger adaptation for the service request we want to simulate.

4.3.2.2 Adaptation Strategy

Table 4.2 summarised the adaptation strategies based on the context elements in table 4.1. In the table some context combination that may trigger specific adaptation are described. The table described context situation where there is no network connectivity.

Strategy1: This describes a situation where there is no network connectivity.

Table 4.2: Summary of the First Adaptation Strategy

Context Element	**Context Range/Value**	**Adaptation Strategy Analysis**
Device Context		
DeviceMemory	• Normal	Not Applicable
Speaker Volume	• Normal	
CPU	• Normal	
Power Level	• Normal	
Environment Context		
Noise		This is a situation whereby there is not network

	• Normal	to connect to the Grid infrastructure to request or deliver services. This network might have been lost due to situation whereby the consumer walks away from where there is strong signal or moving from one cell to another. In this situation, The adaptation strategy is to make the consumer aware of this situation. The utility values for all service variants at this point will be 0.
Available Network	• No Network	

Strategy 2. Table 4.3 described a situation whereby the network currently in use suddenly becomes unavailable but with some other available network.

Table 4.3: Second Adaptation Strategy

Context Element	**Context Range/Value**	**Adaptation Strategy Analysis**
Device Context		
DeviceMemory	• Normal	Other context are normal for service execution
Speaker Volume	• Normal	
CPU	• Normal	

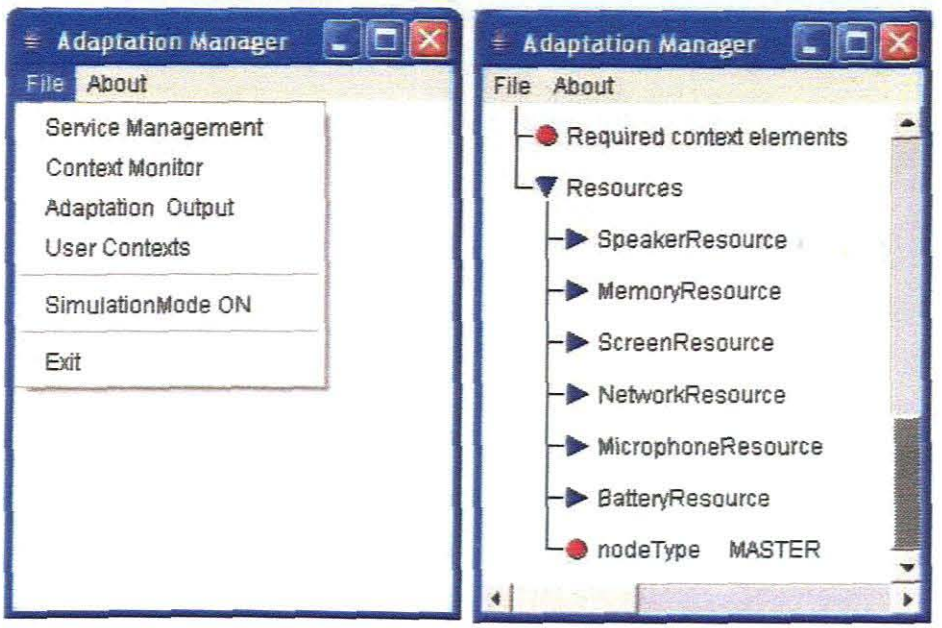

Figure 4.6: Main Implementation Interface and Context Monitor

R3000 model laptop with the following configurations: CPU speed: 3.0 GHz, and 640MB RAM and 60GB hard drive.

4.3.1 Starting the Adaptation Manager

The CACIP adaptation manager when launched, automatically begins context monitoring and values of various contexts and resources monitored are refreshed every 5 seconds. The major parts of the implementation are shown in Figure 4.6. Among these is the Context Monitor that produces all available contexts and device resources with their values as can be seen in Figure 4.6. Service management shown in Figure 4.7 is the menu where currently running service is monitored and managed.

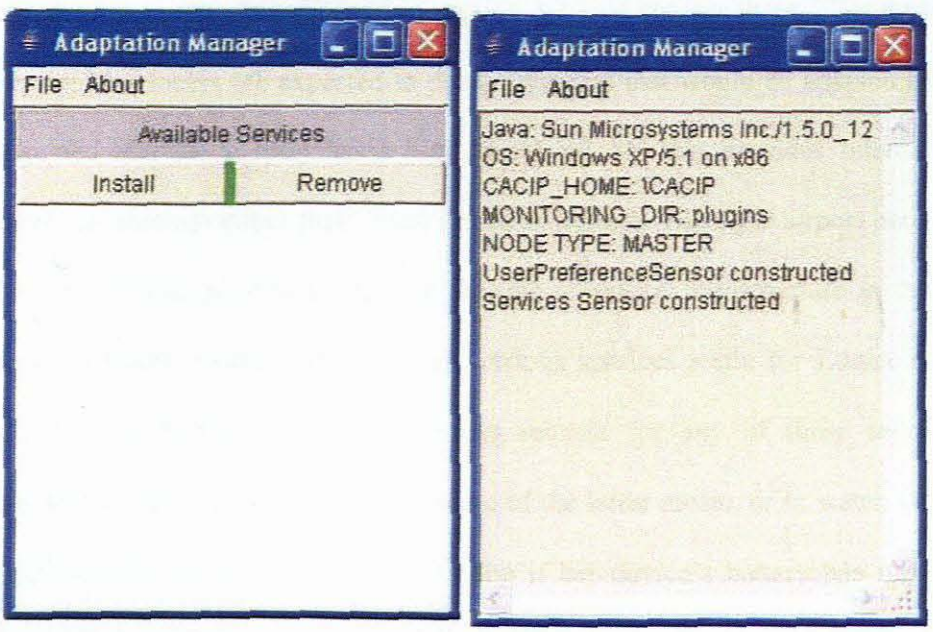

Figure 4.7: Adaptation Output and Service Management Panel

Context Monitor menu is used to observe context that are being monitored and adaptation output is the menu that shows result of the simulated example service is generated. When a service variant is evaluated for selection a line of output is displayed showing its utility and reports if it is selected.

Figure 4.8 also shows the prototype log that shows context and resource changes occurring internally in the system. Once the service is launched, all context sensors for monitoring the service execution contexts are automatically constructed.

4.3.2 Demonstration of the Model

Having presented the detail implementation of the adaptation, demonstration of the usage of this model is now presented. In order to achieve this, a simple hypothetical service request application is simulated. Next this section gives details of the multimedia service request

based on the scenario explained in section 3.2.1 of chapter three. This throws light on how service developers are expected to design services that would be adapted by the model. It is assumed that the Airport has a high speed WLAN that provides Internet connectivity to travellers through either their smart phone or laptops. Also, the airport network is connected to other service providers' services through a service infrastructure in the country. So, as soon as Eunice comes off the plane, various services jostle for Eunice's attention as she walks towards the arrival. Eunice can request for any of these services through her blackberry device for downloading some of the latest music, or to watch latest home videos. She can also transfer these to her laptop if her device's battery has run down or that its memory is not large enough to deliver these music or video services. These two mobile systems have their limitation and requirements as enumerated in chapters two and three.

However, we found that the following additional requirements are more critical to the successful use of Eunice's smart device for service delivery at the airport:

(1) The smart phone must have necessary hardware such as adapter for WMAN, GPRS OR 3G for connectivity to the main airport network and Bluetooth for connectivity of the smart phone with her laptop. Even though they maybe expensive, the GPRS, or 3G network is provided by the Telco companies.

(2) The smart phone must easily pair with the laptop whenever the contexts of execution become unreliable in the smart phone, so that transfer can be made to the laptop with more resources. The communication between the smart phone and the laptop can be established by the Bluetooth.

The assumption here is that the smart phone has the network adapters for connecting to the

infrastructure. To simplify the demonstration, a multimedia service request is simulated for the purpose of evaluating the model as described in the example scenario. The service is assumed to have three main variants that represent three different quality of service in terms of utility that is used to select one of the variants based on prevailing context situation. These variants in reality could be more than three. We decided to use audio, black and white, and full multimedia variants. The audio variant is expected to require less context or resources to execute, followed by the black and white variant and finally by the full multimedia. What follows is a detailed adaptation analysis of the above scenario that is given and this is divided into three main sections.

(1) Context and Adaptation strategy Analysis: A summary of this process is given in subsequent tables that identify the context parameters that influence the adaptation strategies.

(2) A simple component model of the service is given in compliance with our component model described in chapter three.

(3) A simple utility function based design for decision making is also given.

4.3.2.1 Context and Adaptation Strategy Analysis

As illustrated in section 3.3.1.2 of chapter three, three types of context are considered for the purpose of the adaptations envisaged in the scenario. Table 4.1 summarised the context parameters.

(a) Service Context: This describes types of available services or the variants of a given service that service consumer can select from if necessary.

(b) The Device context: This is the context that affects the smart phone such as its memory and processing speed.

Table 4.1:Context Parameter Summary

Context Element	**Context Range/Value**	**Analysis**
Device Context		
DeviceMemory	• Normal • Low	This refers to the amount of device memory available to execute the service. The values normal and low describe the specific value range that this parameter can assume at any given time. This depends on the type of device and its memory.
Speaker Volume	• High • Normal • Low	The speaker volume describes the device speaker which may influence audio variant of the service. The value is assumed to be high or low . This is the same for all devices.
CPU	• Normal • Normal • Low	This describes the processing speed of the device. This varies for different devices. Its value could assume high or low depending on the number currently executing services.
Battery power level	• Normal • Low	This describes the power level of device battery. This may determine whether already initiated service request should be transferred to a consumer's laptop if it's available or the service request be suspended.
Environment Context		

Noise	• High • Normal • Low	This describes one of the user contexts. If the user environment is noisy, this could affect things like the audio and may prompt the system for instance to increase the speaker volume.
Available Network	• GPRS • 3G • WMAN • NoNetwork	This describes all available network types in the user environment whether they are connected or not.
Network Latency	• High • Normal • Low	This describes the network delay which also affects service execution.
Network Bandwidth	• High • Normal • Low	The executing services require some minimum bandwidth to execute. This also may determine what service variants to load for the service consumers
Service Context		
Service availability	• Available • NotAvailable	This describes for instance whether the service is available or not
Request Mode	• Audio • Black and White • Full Multimedia	This describes a situation where service consumer decides what type of this service she prefers.

(c) The Environment context: This is a type that describes possibly the user environment and her needs or choice and also the network contexts.

Power Level	• Normal	
Environment Context		
Noise	• Normal	This is a situation whereby a service request has been initiated and suddenly the network becomes unavailable. However, since other network is available, this is immediately connected. The consumer has to be informed of this context situation. In this case, a 3G network is available.
Available Network	• 3G	

Strategy 3: This is a context situation whereby the available bandwidth of the connected network becomes very low. This is summarised in Table 4.4.

Table 4.4: Third Adaptation Strategy

Context Element	**Context Range/Value**	**Adaptation Strategy Analysis**
Device Context		
DeviceMemory	• Normal	This is a situation whereby the bandwidth of the available and connected network becomes low. This strategy here is to find the service variant with less network bandwidth consumption. For instance the choice will be
Speaker Volume	• Normal	

CPU	• Normal	between audio and black and white variants. However, as soon as the bandwidth becomes normal, the normal service variant which is full multimedia is resumed.
Network Bandwidth	• Low	
Environment Context		
Noise	• Normal	Not Applicable

Strategy 4: This is a situation whereby the battery power becomes very low and this is summarised in Table 4.5.

Table 4. 5: Fourth Adaptation Strategy

Context Element	**Context Range/Value**	**Adaptation Strategy Analysis**
Device Context		
DeviceMemory	• Normal	This is a situation whereby the battery power has become very low in the process of service request. The strategy is to either suspend the service request or transfer it to the consumer's laptop if it's available.
Speaker Volume	• Normal	
CPU	• Normal	

Battery power level	• Low	
Environment Context		
Noise	• Normal	Not Applicable
Available Network	• GPRS • 3G • WMAN • Others	

In summary, the list is not exhaustive; however, this gives a picture of how the context and adaptation strategies are designed for all services that can be adapted by the model.

What follows, is a description of the component model of the service which is based on the components model described in section 3.4.1 of chapter three.

4.3.2.3 Service Component Architecture

We start with the component types. There are four major component types.

(1) The Component Administrator: This serves such functions such as:

(a) Connecting and disconnecting other service components during execution,

(b) Keeps current state of the service,

(c) Facilitates component communication and

(d) May also perform some component management functions.

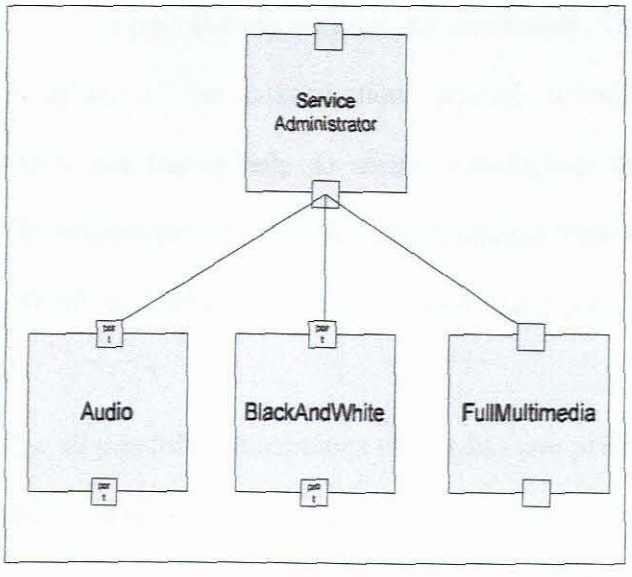

Figure 4.8: Components of the Multimedia Service Example

(2) The audio variant component represents the audio variant of the service which is an atomic component of the service composite component.

(3) Black and White Component: This is basically the same as above but represents the black and white variant of the service

(4) Full Multimedia: This is also similar to the one above but represents the full multimedia variant of the service.

Figure 4.9 illustrates these component types and their relationship using their connectors and ports.

This figure illustrates a high level view of the simulated service request components. Each of the service variants: audio, black and white and full multimedia has its own administrator namely a*udioAdmin*, *BnWAdmin* and *FullAdmin*.

4.3.2.4 Specification of the Composition Plans and Architectural Constraints

The component model of the service presented in section 4.6.2 above is not complete without

the composition plan and the architectural constraints. The compositional plan helps to store the description of the collaborations among several service components, while the architectural constraints help to avoid meaningless service variants instantiation when resolving variation points of the service component framework. The component types such as Audio, Black and White may have several realisations which are dynamically created at runtime.

In theory, all possible combinations of selected components at the variation points represent valid service variants. However, in practice, this selection maybe meaningless as the selected components may not form a feasible service variant in terms of their provided and required properties. Therefore, the composition plan and the architectural constraints help to avoid these situations. For instance, we will not want components that form Black and White variant to be combined with those of full Multimedia components. Hence, the need to create the composition plans for this composition along with the architectural constraints. The architectural constraints represent the connectors and ports that can be connected or those that cannot be connected to form meaningful service. The concept of feature which is borrowed from MADAM [25] is used to keep various components variations of the multimedia service. This is necessary to avoid meaningless instantiation of service components that form a given service variant at runtime.

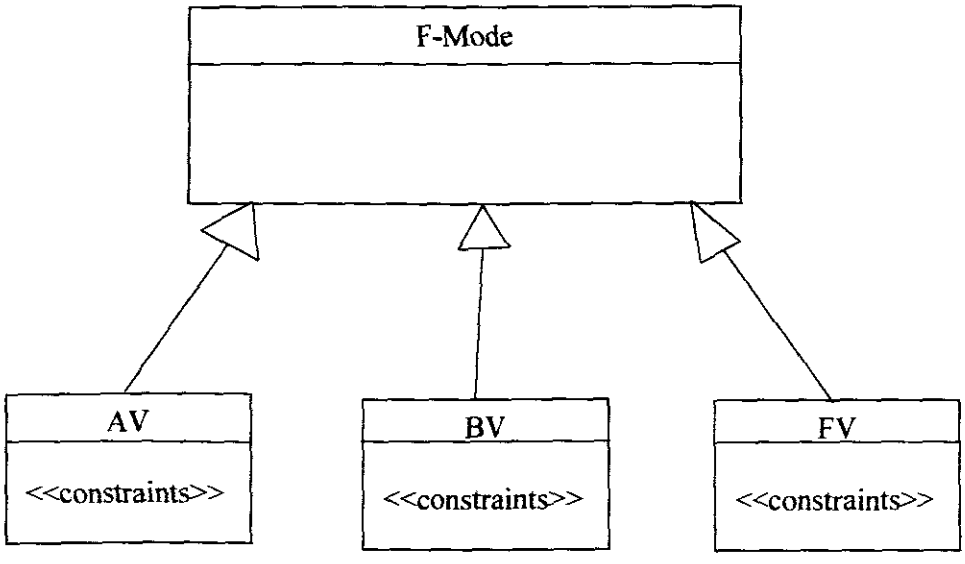

Figure 4.9: Variation Point Stored in F-Mode

So the component that keeps information about various components of the multimedia service, here is called F-Mode, which consequently has three subcomponents namely AV BV and FV representing Audio, Black and White and Full Multimedia variants respectively. This F-Mode keeps the variations and architectural constraints of the components of the service as illustrated in Figure 4.10.

Following the presentation of the components of the service and the corresponding component plan, we present in the following lines, how those components are associated with properties that characterise their extra-functional behaviours. Context analysis presented earlier helps to associate these properties with each of these components. Table 4.7 summarised this procedure.

Finally, as the component properties of the adaptive service model have been annotated, the implementation of the utility function algorithm is presented.

Table 4.6:Summary of utilities for Context parameters

Property Type	Abbreviation	Analysis
Available Mode	AvailMode	This property defines the offered service modes. It can take the following values. • AV – Audio Variant • BV- Black and White Variant • FV- Full multimedia Variant The mode preferred by the service consumers is given by the context element RequiredMode whose value is compared with the AvailMode in the evaluation of the utility function value. The service consumer will have higher weight during utility value calculation.
Response Time	ResponseTime	This property is related to the service quality and it represents the maximum response time required by a component to work correctly. It is measured in milliseconds. The full multimedia needs better network condition than the audio variant. This implies that Response time value for audio will be lower than that required by the Black and White variant. In the utility function calculation, the value of Response Time is compared with Response Time offered by the environment. From this property, service variant that do not work correctly as a result of bad network condition, even if preferred by the service consumer is discarded.
Sound level	AvailSound	This is an integer value that represents the sound level offered by those components that emit sound. In this case all our component emit sound and the value will depend directly on the speaker volume which is a setting to be provided by the service developers. In the utility function, a speaker configuration that improves the operation of the service in a noisy environment such as in an airport environment will be selected.
Power Factor	ConsumedPower	This provides information about the power consumption of the service variant. For this service, it's a function of the selected network segment or type. This is a device setting to be provided by the service developers. For example. a GPRS network consumes less power than a WiFi network. So *ConsumedPower* by *GPRS* < *ConsumedPower* by WiFi. Hence if *ConsumedPower* by GPRS is 3 and that of a WiFi is 6, then service component variant associated with GPRS will be preferred to service variant component associated with WiFi network as this helps to predict time activity of these service variant components.
Memory Required	MemoReq	This stores the amount of memory required by each component for it to work correctly.

```
If(!AVRequested and AvailMode =
"AV"){
    Return 0;
}else
 If(!BVRequested and AvailMode =
"BV"){
    Return 0;
}else
If(!FVRequestedAndAvailMode
="FV"){
    Return 0;
}Else
If(!NetworkSegRequestedAnd
NetworkSeg = "None"){
    Return 0;
}
Else
If(!NetworkAvailability(networkUsed)){
    Return 0;
} else
If(AvailMode = RequestedMode){
    UtilUserRequest = 1;
Else {
    UtilUserRequest = 0;
} else
if (AVRequested and AVAvail)
{
    UtilAV = 1;
    UtilBV = 1;
Else{
    UtilBV=0;
}
If(OfferredResponseTime<Response
TimeRequested{
    UtilResponse =1;
}else
{
    UtilResponse =0;
}
Utility = w1*UtulUserRequest
-w2*UtilAV + w3*UtilAV
-w4*UtilBV + w5*UtilResponse);
```

Figure 4.10: Utility Function Algorithm Pseudo Code

Legend

AV = Audio Variant

BV = Black and White Variant

FV= Full Multimedia variant
AVRequested, *BVRequested* and *FVRequested* represent requested variants respectively

UtilUserRequest = Utility for user Request

UtilAV = Utility for Audio Variant

UtilBV = Utility for Black and White variant

UtulFV = Utility for Full Multimedia variant

UtilResponse = Utility for Response Time

4.3.2.5 Algorithm Implementation

The objectives of this algorithm are (1) To ease the implementation of the utility function model and (2) to draft how the utility function is created. The number of service variants of a given service determines the number of times the utility function algorithm will be executed during adaptation process.

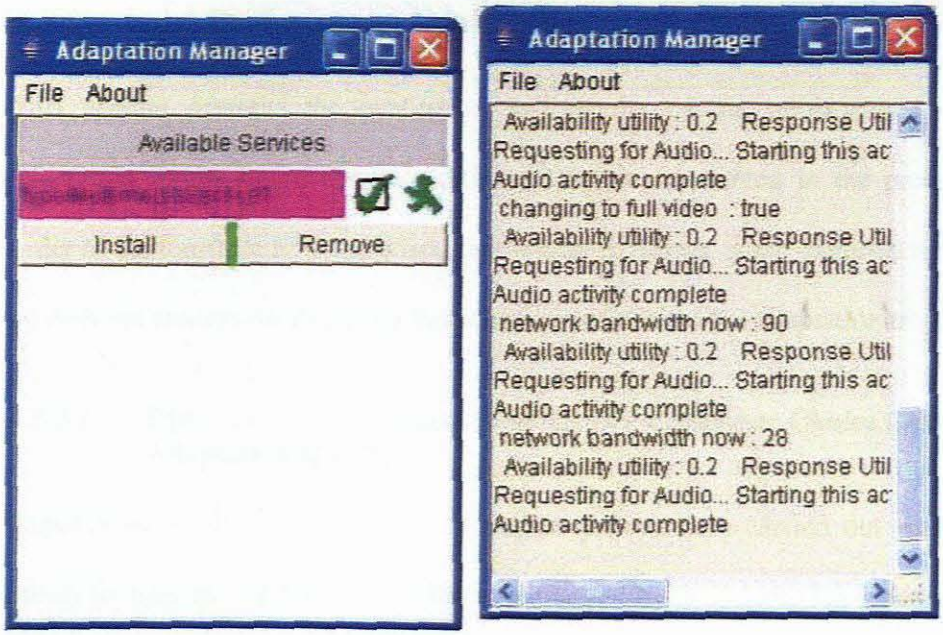

Figure 4.11: Launching a Service and Adaptation Output

This means that in the implementation, variant with required context utility that does not meet the available context utility is rejected first. In the remaining part of the evaluation, different properties that are relevant in the decision making such as consumer's preference, network condition or the device memory are then evaluated. For each property, a partial utility is calculated, and the final utility is the weighted sum of the partial values according to the utility function model presented in section 3.3.2.2 of chapter three. The pseudo code in Figure 4.11 below gives a glimpse of the utility function.

The service is implemented in java and the jar file was ported into the adaptation prototype and Figure 4.12 illustrates output generated during the execution of the service in the adaptation manager prototype.

4.3.3 Model Performance Evaluation

This section presents the evaluation and results of the performance of the proposed adaptation model based on the multimedia service presented in the preceding section. In order to demonstrate how the adaptation model performs, some experiments were conducted on how the context-aware utility-based model influenced the adaptation triggering.

4.3.3.1 Effect of Service Variants and Service Consumer Choice (Weight) on Adaptation Quality

Experiment on the quality of the adaptation process was carried out which basically was meant to measure the level of satisfaction attained by the adaptation process as perceived by the service consumers. Two factors influenced the choice of metrics we used to measure the quality of adaptation. First, the service variant selected by the adaptation model is expected to satisfy the execution contexts requirements. This means that the utilities for both required and available context parameter values must be feasible in order to have a service variant presented to the service consumers. Second, the choice of variant the service consumer makes also needs to be satisfied in adverse context situation. In other words, it means that the more the service consumer is satisfied with a service she is presented with, the better the adaptation. Adaptation Quality Index (AQI): is defined as the measure of quality the adaptation model achieves as perceived by the service consumers. This is calculated using the following parameters.

Context Utility CI: This is calculated from the *RequiredContextUtility* and the *AvailableContextUtility* given as:

Table 4.7: Experimental Results

Weight	No of Variants	Context Utilities	IAQI	Adaptation Time	Response Time without Adaptation	Response Time
0.9	10	0.0588734	2.0301936	31	94	125
	20	0.061008	0.90557039	62	110	172
	30	0.0620973	0.72924334	93	125	218
	40	0.0592659	0.5254227	125	141	266
	50	0.0562116	0.39442417	156	156	312
	60	0.0580518	0.34486202	204	203	407
	70	0.0586652	0.29726798	234	250	484
	80	0.0595746	0.25794452	281	250	531
	90	0.0594717	0.22433771	297	265	562
	100	0.0582682	0.20593692	344	390	734
0.8	10	0.0689471	2.10433695	32	78	110
	20	0.0627557	0.94061866	47	62	109
	30	0.0669702	0.71260961	94	125	219
	40	0.0657288	0.51441572	109	156	265
	50	0.0670764	0.42509933	141	172	313
	60	0.0663065	0.3357652	172	125	297
	70	0.0672196	0.30232961	203	219	422
	80	0.0660413	0.25056914	234	172	406
	90	0.0663507	0.23093067	266	281	547
	100	0.0656323	0.19865548	296	329	625
0.7	10	0.0760788	2.11153952	31	16	47
	20	0.0768899	1.11806309	46	32	78
	30	0.0771993	0.72466693	78	125	203
	40	0.0784578	0.53562651	94	78	172
	50	0.0773772	0.48563313	125	125	250
	60	0.0797776	0.36164135	156	406	562
	70	0.0782335	0.30169393	203	235	438
	80	0.0788941	0.27069791	219	250	469
	90	0.0781413	0.23153365	250	250	500
	100	0.076044	0.21275054	235	327	562

$$CI = \frac{RequiredContextUtility}{AvailableContextUtility} \quad (11)$$

Adaptation Quality:

$$CI = \frac{Rcu}{ACu} \quad (12)$$

Where R_{CU} denotes the required context utility and A_{CU} denotes the Available Context Utility

$$\text{Service Consumer choice Index SI} = \frac{Total\ Choice\ satisfied}{Total\ choices\ made} \quad (13)$$

A higher value of this parameter index implies that the selected service variant is more important to the service consumer.

$$\text{Finally, } QAI = (CI + SI)/2 \quad (average\ of\ CI\ and\ SI) \quad (14)$$

It is assumed that the two factors considered in the experiment carry equal weight and this weight is adjusted according to how much importance is attached to each of these factors.

In order to measure AQI, the number of service variants for a given service request and the weight were varied. The weight is a numeric value that determines service consumer's choice of service variant. The higher the weight for a given variant, the more the consumer prefers that variant. The Bar chart plot of AQI against service variants for weights 0.9, 0.8, 0.7 is shown in Figure 4.12.

It can be inferred from figure 4.12 that the more relaxed the service consumer choice in terms of weight, the better the quality, but this degrades as the number of service variants to select from increased. This shows that when the weight was 0.7, there was better adaptation quality of 2.11 compared with that of weights 0.9 and 0.8 which were 2.03 and 2.10 respectively when the number of service variants was 10.

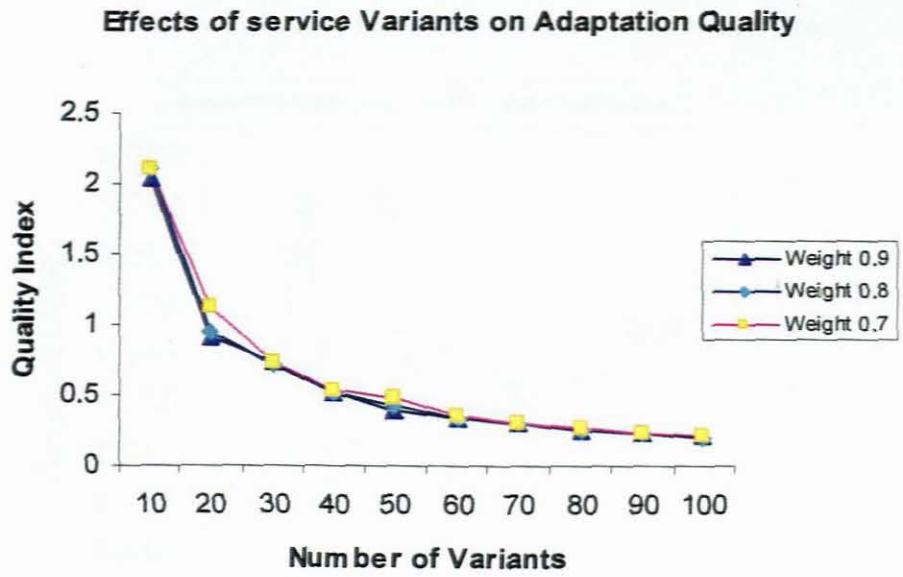

Figure 4.12: Effect of Service Consumer Choice and Service Variant On Adaptation Quality

4.3.3.2 Effect of Adaptation on Overall Response Time

Another experiment was conducted to evaluate the effect of adaptation on Response Time. Two metrics were used; the context utility and response time. The context utility is defined as the ratio of the required context to that of available context. The response time is defined as the time in milliseconds between when a request for a given service was made and when it was delivered.

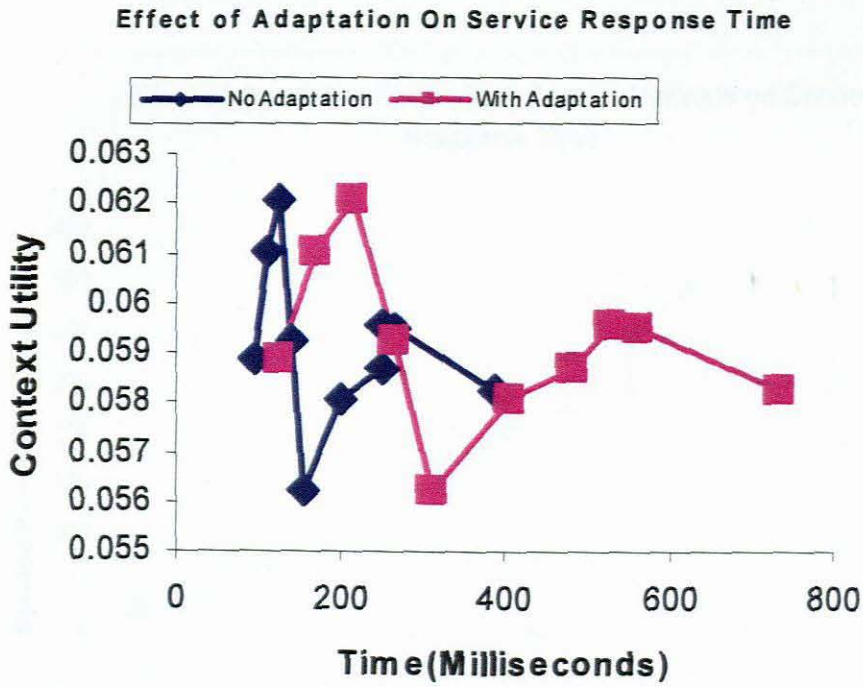

Figure 4.13: Effect of Adaptation on Service Response Time

During the experiment, the response time was measured between when a service request was made and when a response from the system was received. Two versions of this experiment were conducted. One was when the model was used to adapt the service and the other was conducted without adaptation process. The experiment was carried out over some 30 minutes of simulation time; the context utilities and corresponding service response time were measured and observed behaviour was plotted as shown in Figure 4.13. We observed that the context utilities did not show any trend but that it kept changing sometimes with increasing and some other time decreasing.

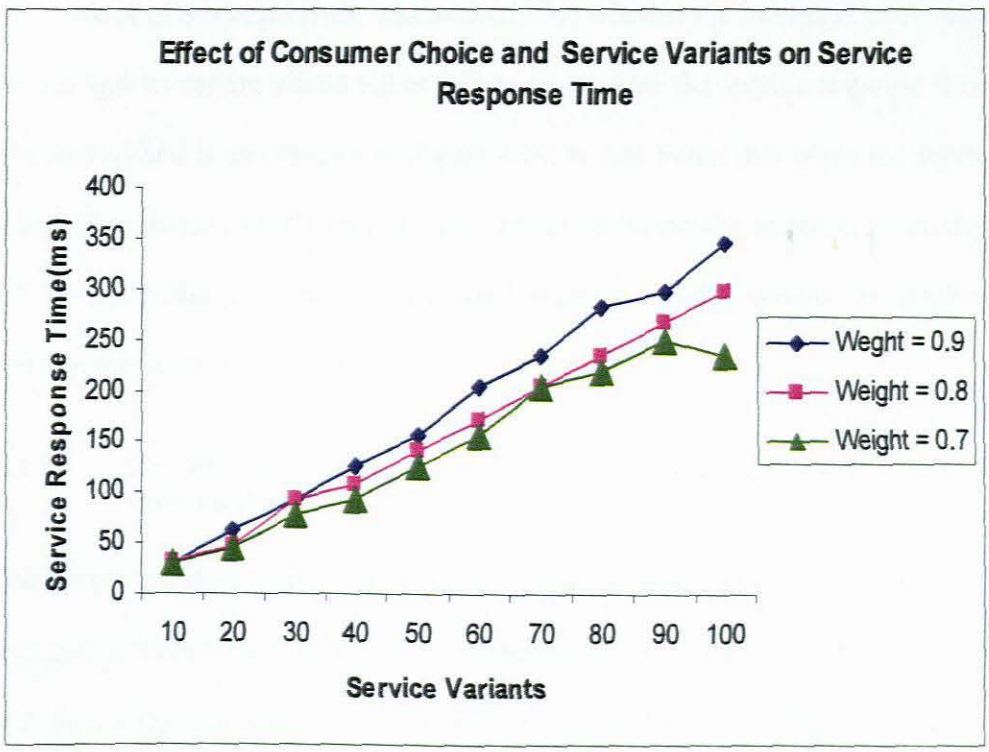

Figure 4.14: Effect of Service Consumer Choice and Service Variants on Service Response Time

This is very typical of the dynamic instability of the environmental contexts which cannot be predicted. The plot shows that with time, the response time for both adaptation series and none-adaptation series increased. However, the rate at which the response time increased for adaptation series is lower compared with non- adaptation series.

4.3.3.3 Effect of Service Consumer Choice and Service Variants on Service Response Time

A separate experiment was conducted to ascertain the impact of service consumer choice and number of service variants on the service response time. In the experiment, we varied the number of service variants and the service consumer choice which was measured in terms of weights. The corresponding service response time was determined for three different

weights 9, 8, 7, respectively, and number of service variants varied between 10 and 100.

The essence of this experiment was to determine whether the increased preference for a given service and its variant would either reduce or increase the service response time. The result was plotted and is as depicted in Figure 4.14. It was found that when the service consumer relaxed her choice (weight), the system performed better. For instance, when the weight was 0.7, though initial response time increased, it began to fall compared to weights 0.9 and 0.8 which rather kept climbing up.

4.3.3.4 Comparing Adaptation time with Response time as number of variants Increased

The third experiment was to evaluate the effect of consumer choice and service variants on response and adaptation times. In the experiment, we varied the number of service variants and then subjected the service request to adaptation. The time it took to evaluate and reconfigure the service was measured; the overall response time was also measured.

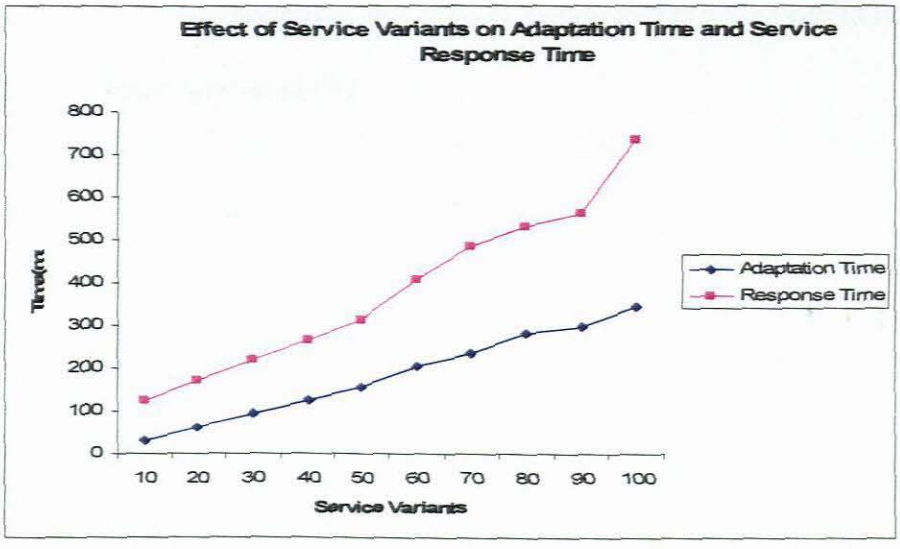

Figure 4.15: Effect of Service Variants on Adaptation Time and Service Response Time

The adaptation time, that is, the time it takes to evaluate the service variants is compared with the overall response time which is the time between when a request for a service is received and when a response is received or when the service is delivered. Figure 4.15 reveals that as the number of service variants increased, the adaptation time and consequently the response time increased. It was observed that as we increased the number of service variants, though both response and adaptation time increased the adaptation time was steady. However, it was observed that the response time increased at higher rate than the adaptation time.

4.3.4 Conclusion

In this chapter, the implementation and simulation of the proposed context-aware utility function based adaptation model were presented as a demonstration. A performance evaluation of the model showed that adaptation is very useful to provide considerable service quality as experienced by service consumer through series of experiments that have been

conducted. The experiments also showed clearly that there is need to improve the model to provide better service quality.

CHAPTER FIVE

CONCLUSION AND FUTURE WORK

5.1 Overview

This chapter concludes the dissertation with a summary of the investigation and the overview of the contributions. A brief discussion of some of the open research questions unearthed in the course of this investigation is then presented.

5.2 Conclusions

Adaptable Mobile Grid services should be able to modify its own behaviors at run time to adjust to changes in its execution contexts in order to optimize its performance and availability in a transient environment. However, traditional services (be it Grid or Web) performed their functions with explicit inputs and are not aware of their contexts. Access to context information, therefore, provides an opportunity for using contexts in service provisioning that may result in higher quality of service. This context can be used to reason about adjustments the systems may have to make in response to changes in contexts that may otherwise negatively impact on the quality of service delivery. Making services to reason and modify their behaviors at runtime is a complex task that has been addressed in a number of ways; from content adaptation to user interface adaptation even service function has been adapted. All these adaptation mechanisms and strategies are yet to provide needed service quality in terms of service response time which is still very high as a result of the dynamic nature of mobile technology.

This work attempted to fill the gap found in the mobile Grid based E-service infrastructure proposed in the centre. The e-Service technology is based on the emerging concept of utility grid computing. In the GUISET infrastructure, the interaction between the mobile client components and the server components posed serious challenges for two reasons. First, the wireless network connecting these components is unstable. Second, the heterogeneity of the mobile devices poses interaction problem. In depth analysis and study of this interaction problem led to the discovery of the increased response time between a service request from the mobile clients and when the server component (service providers) deliver the requested services.

In order to address this problem, context-aware adaptation of service component interaction was proposed in order to improve on the quality or satisfaction experienced by a mobile grid service consumer. The study was founded on a methodical definition of interaction, and then followed by fitting the existing interaction pattern (CACIP) into mobile grid setting, and finally the birth of a context-aware utility function based adaptation model. A dynamic utility function based decision algorithm was also proposed. All these served as the objectives to achieving the main goal of proposing a model for dynamic and adaptable system of service interaction in mobile grid. Context-aware in the model meets the need of the running application services to be aware of the platforms and environment on which they are running. Context-aware services are enabled to take both reactive and proactive decision on what to do in case the contextual situation does not provide feasible execution context for their hitch-free execution. This aspect was handled by the context monitor of the model. Furthermore, there was need for the model to reason about monitored contexts in order to decide at runtime

what to do should the context fluctuate adversely. We designed a utility based decision algorithm which decides on feasible adaptation options.

The algorithm works on the assumption that services are designed by the developers to have alternative implementation called variants. A variant differs in context requirements. The crafted model selects at runtime which would be feasible under the current prevailing context situations.

To effect adaptation, service reconfiguration pattern was adopted. The pattern is based on component technology, executes adaptation decision reached by the Evaluator module. The choice of this pattern was based on the fact that it is component based and facilitates easy composition of services not only at design time but at runtime. The proposed model was implemented with the integration of MADAM APIs and libraries [25] for context management, and component reconfiguration.

These design fulfilled the goal and the objectives that were set for this investigation to achieve as a simple service request was simulated in the framework to evaluate the proposed model. A number of experiments were conducted to evaluate and to prove the concepts adopted in the model design. One of the experiments was to determine the effect of number of service variants and service consumer choice on the quality of adaptation which was defined as the satisfaction or quality perceived by the service consumers. As expected, it was observed that the more the number of variants, with increasing choice of service variants (measured by weights), the more the adaptation quality degraded.

Another experiment was conducted to determine the effect of adaptation process on service response time. In this experiment, runtime context utilities were plotted against the response time measured when adaptation process was executed and when it was not applied. We found

out that the response time with adaptation is lower than that without adaptation at any given context utility. This is very significant as it proved that adaptation can actually help reduce response time. The third experiment was to evaluate the effect of consumer choice and service variants on response and adaptation times. In the experiment, we varied the number of service variants and then subjected the service request to adaptation. The time it took to evaluate and reconfigure the service was measured. Also, the overall response time was measured as we varied the weights. The result showed that as we increased number of service variants, though both response and adaptation time increased the adaptation time was steady.

Finally, the last experiment was to evaluate the effect of weights or service consumer choice on response time, the result showed that when the consumer choice was relaxed, the response time was not as high as when the choice was high as determined by the applied weight.

5.3 Future Work

The evaluation results of the proposed model unearthed some important questions that formed the basis for future investigations. For example, in the first experiment, the adaptation quality degraded faster with increasing number of variants and with high service consumer choice. This means that if a service consumer insists on having a given service variant at the expense of others, the quality experienced by such service consumers is very poor. There is need to look critically into the utility-based model and decision algorithm to improve on giving priority to service consumers choice and integrating this in to the model so as to improve on the overall quality experienced by the service consumers.

Also, it was observed that there was very high disparity between the adaptation time and the overall response time. The cause of this is yet to be ascertained but this point to the fact that

there is some negative effect of service reconfiguration which time did not allow us to investigate in this research. Hence, there is need to investigate message exchanges among the components of the adaptation model. Standard protocols for communication, for "*eventing*", messaging and addressing need to be looked into. Also, to facilitate runtime binding for structured information exchange between adaptation elements, the interfaces and model have to be defined in machine readable manner.

Another aspect of this work that needs further investigation is the integration of the adaptation model into practical grid infrastructure as concluded in [81]. This we think can be done by integrating adaptation as service that can be consumed by both service consumers and providers guided by the service oriented architecture standards. This means that rather than embedding adaptation techniques into the grid application services, it can be provided as a service for the purpose of reusability and interoperability. It means that service developers will not bother about designing adaptation in the grid application services.

REFERENCES

[1] Gaddah A, and Kunz T., " A survey of middleware paradigms for mobile computing", Carleton University, Systems and Computing Engineering,2003 Technical Report (SCE-03):Available from: http://www.sce.carleton.ca/wmc/middleware/middleware.pdf(accessed 28th June (2006)

[2] Katz R., "Adaptation and Mobility in Wireless Information Systems", IEEE Personal Communication, Vol. 1:pp6-17, 1994.

[3] Chu D., and Humphrey M., "Mobile OGSI.NET: Grid Computing on Mobile Devices", 5th IEEE/ACM International Workshop on Grid Computing - Grid2004 (at Supercomputing 2004). Nov 8 2004, Pittsburgh, PA.

[4] Guan T., Zaaluska E. and Roure D." A Grid Service Infrastructure for mobile Devices", Proceedings of the First International Conference on Semantics, Knowledge and Grid, 2006.Available from:http://doi.ieeecomputersociety.org/10.1109/SKG.2005.10

[5] Zuma S., and Adigun M.,"CACIP: Context Aware Components Interfacing Pattern", In 2006 Proceedings IASTED Conference of Modelling and Simulation: Available from http://www.actapress.com/PaperInfo.aspx?PaperID=26815006 [Accessed 29th June 2006]

[6] Adigun M.O., "Software Infrastructure for e-Commerce and e-Business" unpublished Research Working paper, Res-CSD-01, Centre for Mobile e-Services, University of Zululand, 2006

[7] Chaari T., Laforest F. and Celentano A. "Adaptation in Context Aware Pervasive Information Systems" Journal of Pervasive Computing And Communication Vol. 2 No. 2, June, 2006

[8] Kistler J, and Satyanarayanan M. ,"Disconnected Operation in the Coda File System". ACM Transactions on Computer Systems, vol. 10, no. 1, February.1992.

[9] Foster, I., Kesselman, C., Nick, J. and Tuecke, S., "The Physiology of the Grid: An open Grid services Architecture for Distributed Systems" Available from:

http://www.globus.org/alliance/publications/papers/ogsa.pdf.

[8] Goyal S. and Carter J., "A Lightweight Secure Cyber Foraging Infrastructure for Resource-Constrained Devices". Appears in Sixth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA2004), pp. 186-195.

[11] Bruno D., Scarpa M., Zaia A., and Puliafito A. "Communication Paradigms for Mobile Grids Users", 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, 2003, Available from: http://ieeexplore.ieee.org/Xplore/login.jsp?url=/iel5/8544/27003/01199431.pdf

[12] Kola G. , Kosar T. and Livny M., " Run Time Adaptation of Grid Data Placement Jobs",Computer Sciences Department, University of Winconsin Madison, USA, 2004:Available from: http://www.cs.wisc.edu/condor/stork/papers/runtime_adaptation-pdcp2004.pdf

[13] Al-bar A. and Wakeman I., "A survey of Adaptive Applications", Proceedings of the 21st International Conference on Distributed Computing Systems Workshops", 2001. Available from: http://ieeexplore.ieee.org/iel5/7338/19870/00918713.pdf

[14] Bradram J."The Java Context Awareness Framework (JCAF)-A Service Infrastructure and Programming Framework for Context Aware Applications Tutorial", Available from:http://www.daimi.au.dk/~bardram/jcaf/jcaf.v15.pdf

[15] GT4: www.globus.org/toolkit/

[16] OSGi: The OSGi Service Platform - Dynamic services for networked devices. Available from:http://www.osgi.org/ [accessed: June 23, 2007].

[17] OWL-S: www.w3.org/Submission/OWL-S/

[18] Rossi P. and .Ryan C. ,"Emperical Evaluation of a Local Adaptation Algorithm",RMIT University, Melbourne,Australia,2005.Available from: http://goanna.cs.rmit.edu.au/~caspar/ATcrc/1.2/papers/DOAPaper2005.pdf

[19] Aksit M. and Choukair Z. ,"Dynamic, Adaptive and Reconfiguration Systems Overview and Prospective Vision", Proceedings of the 23rd International Conference

on Distributed Computing Systems Workshop,2003.Availablefrom: http://ieeexplore.ieee.org/Xplore/login.jsp?url=/iel5/8560/27094/01203537.pdf

[20] Kurkovsky S., Bhagyavati A. R., M.Yang.'Modeling a Grid-Based Problem Solving Environment for Mobile Devices', In Proceedings of the InternationalConference on Information Technology: Coding and Computing (ITCC'04), Las Vegas, Nevada, April 05 - 07, 2004.

[21] Mikic-Rakic M. and Medvidovic N. "Support for Disconnected Operation via Architectural Reconfiguration", International Conference on Autonomic Computing (ICAC'04) New York, May, 2004.

[22] Aksit M. and Choukair Z."Dynamic, Adaptive and Reconfigurable Systems Overview and Perspective Vision", Proceedings of the 23 rd IEEE International Conference on Distributed Computing Systems Workshops, 2003

[23] Geighs K., Khan M.U, Reichle R. Solberg A., Hallsteinsen S. and Merral ."Modelling of Component based Adaptive Distributed Applications", Proceedings of the ACM 21st Symposium on Applied Computing,2006. Bourgogne University, Dijon, France.

[24] Buisson J., Andre F. and Pazat J. "Dynamic Adaptation for Grid Computing"INRISA/INSA, Rennes, France. Available from:http://www.irisa.fr/paris/Biblio/Paper/BuiAndPaz05EGC.pdf, Accessed: 5th March, 2007

[25] MADAM Deliverables: Available from: www.istmadam.org, accesed last 25th April, 2007.

[26] Dey A. "Providing Architectural Support for Building Context-Aware Applications". PhD thesis. College of Computing, Georgia Institute of Technology. 2000.

[27] Keeney J., and Cahill V. "Chisel: A policy-driven, context-aware, dynamic adaptation framework". In proceedings of the 4th International Workshop on Policies for Distributed Systems and Networks. 2003. Lake Como, Italy.

[28] Walsh W.E, Tesauro G., Kephart J.O and Das R. "Utility Functions in Autonomic Systems". In proceedings of First International Conference on Autonomic Computing (ICAC'04). 2004. p. 70-77.

Systems". In proceedings of First International Conference on Autonomic Computing (ICAC'04). 2004. p. 70-77.

[29] Simula Research Laboratory. The QuA project. Available From:http://bagadjimbiri.simula.no:8888/QuA [accessed: May 23, 2007].

[30] McKinley, P.K., Sadjadi S.M., Kasten E.P. and Cheng B.H."A taxonomy of Compositional Adaptation". Tech report, Software Engineering and Network Systems Laboratory, Michigan state university. 2004. pp 56-64

[31] Ranganathan A. and Campbell R.H." An infrastructure for context-awareness based on first order logic" International Journal of Personal and Ubiquitous Computing , Vol 7 Number 6, 2003, Springer Verlag London pp 353-364

[32] University of Illinois at Urbana-Champaign. The Illinois GRACE Project (Global Resource Adaptation through Cooperation). Available from: http://rsim.cs.uiuc.edu/grace/index.html[accessed: June 24, 2007].

[33] Adelstein, F., Gupta S.K., Richard G. and Schwiebert L.. Fundamentals of Mobile and Pervasive Computing. 2005. McGraw-Hill Professional. ISBN 0071412379.

[34] Akbar, M.M., Manning E.G., Shoja G.C. and Khan S. Heuristic Solutions for the Multiple-Choice Multi-Dimension Knapsack Problem. In proceedings of International Conference International Conference on Computational Science. 2001. San Francisco, USA.

[35] Hifi, M., M. Michrafy, and A. Sbihi. Heuristic algorithms for the multiple-choice multidimensional knapsack problem. Journal of the Operational Research Society, 2004.55(12): p. 1323-1332.

[36] Khan, S., Li K.F., Manning E.G. and Akbar M.D. "Solving the Knapsack Problem for Adaptive Multimedia System. Studia Informatica, Special Issue on Cutting, Packing and Knapsack Problems, 2002. 2(1): p. 155-177.

[37] Sousa, J.P., Poladian V., Garlan D., Schmerl B and Shaw M. "Task-Based Adaptation for Ubiquitous Computing". IEEE Trans. on Systems, Man, and Cybernetics, Part C: Applications and Reviews, 2006. 36(3): p. 328- 340.

[38] Belaramani N.M "A Component-based Software System With Functionality Adaptation for Mobile Computing". Master's thesis, The University of Hong Kong, 2002.

[39] Pillai P. and Shin K. G. "Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems". In Proceedings of the 18th ACM Symposium on Operating System Principles, pp. 89-102, October 2001.

[40] Han R., Bhagwat P., LaMaire R., Mummert T., Perret V. and Rubas J. Dynamic Adaptation in an Image Transcoding Proxy for Mobile Web Browsing. In IEEE Personal Communication, vol. 5(6), pp.8-17, 1998.

[41] Gross T., Steenkiste P. and Subhlok J. "Adaptive Distributed Applications on Heterogeneous Networks". In Proceedings of the 8th Heterogeneous ComputingWorkshop, pp.209-218, April 1999.

[42] Fox A., Gribble S., Chawathe Y. and Brewer E. A. "Adapting to Network and Client Variation UsingActive Proxies Lessons and Perspectives". In IEEE Personal Communications, Special Issue onAdaptation, August 1998.

[43] Flinn J.and Satyanarayanan M. "Energy-aware Adaptation for Mobile Applications". In Proceedings of the 17th ACM Symposium on Operating System Principles, pp. 48-63, December 1999.

[44] Corradi A., Montanari R. and Stefanelli C. "How to Support Adaptive Mobile Applications". InProceedings of WOA2001, September 2001.

[45] Chandra S., Ellis C. S., and Vahdat A." Mulitmedia Web Services for Mobile Clients Using QualityAware Transcoding". In Proceedings of the Second ACM International Workshop on Wireless MobileMultimedia(WOWMOM '99), pp. 99-108, August 1999.

[46] France, R., Ray I., Georg G. and Ghosh S.. An aspect-oriented approach to design modeling. IEE Proceedings -Software, Special Issue on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design. 2004. pp 151(41).

[47] Otebolaku A.M., Adigun M.O., Iyilade J.S. and Ekabua O.O., "On Modelling Adaptation in Context-Aware Mobile Grid Systems," icas, Third International Conference on Autonomic and Autonomous Systems (ICAS'07), 2007 p. 52, .

[48] Chen, G. and Kotz D. "A survey of context-aware mobile computing research". Department of Computer Science, Dartmouth College, Dartmouth. 2000. Technical report TR2000-381.

[49] Sousa J.P. and Garlan D. "Aura: An Architectural Framework for User Mobility in ubiquitous Computing Environments". In Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture, 2002. pp. 29-43

[50] Alia M., Wold Eide V.S.,Paspallis N.,Hallsteinsen .O.,Papadopoulos G.A. "A Component-based Planning Framework for Adaptive Systems". In proceedings of 8th International Symposium on Distributed Objects and Applications (DOA). 2006. Montpellier, France. Springer Verlag.

[51] Huebscher, M.C. and J.A. McCann. "An adaptive middleware framework for context-aware applications". Personal and Ubiquitous Computing, 2006. 10(1): p. 12-20.

[52] Alia M., Wold Eide V.S., Paspallis N., Eliassen F., Hallsteinsen S.O., Papadopoulos G.A " A Utility-Based Adaptivity Model for Mobile Applications". AINA Workshops (2) 2007: pp556-563.

[53] Want R.,Hopper A., Falcao V., and Gibbons J. "The Active Badge location system". ACM Transactions on Information Systems, 1992. 10(1): p. 91-102.

[54] Foreman G.,and Zahorjan J. "The Challenges of Mobile Computing", IEEE Computer, April 1994, pp. 38-47.

[55] Banavar G. and Bernstein A.," Challenges in design and software infrastructure for ubiquitous computing applications". Advances in Computers 62: pp180-203 (2004)

[56] Liu H., Bhat V., Parashar M. and Klasky S., "An Autonomic Service Architecture for Self-Managing Grid Applications," Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing (Grid2005), Seattle, WA, USA, IEEE Computer Society Press, November 2005.

[57] Rasche, A. and Polze A. "Configurable Services for Mobile Users". In proceedings of Seventh IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2002). 2002. San Diego, California, USA. p. 163.

[58] Parashar M. and Pierson J.M., "When the Grid becomes Pervasive: A Vision for Pervasive Grids," Research Position Paper, 2007 Available from: http://www.caip.rutgers.edu/TASSL/ [Accessed: 1st October 2007]

[59] Parashar M. and Hariri S., "Autonomic Grid Computing – Concepts, Requirements, Infrastructures," "Autonomic Computing: Concepts, Infrastructure and Applications," Editors: M. Parashar and S. Hariri, CRC Press, 2006.

[60] Ryan N., ConteXtML: Exchanging Contextual Information between a Mobile Client and theFieldNote Server. Available from:http://www.cs.kent.ac.uk/projects/mobicomp/fnc/ConteXtML.html [accessed: August, 2006].

[61] Voelker, G.M. and B.N. Bershad. Mobisaic, An Information System for a Mobile Wireless Computing Environment & Engineering In proceedings of IEEE Workshop on Mobile Computing Systems and Applications. 1994. Santa Cruz, CA, US.

[62] Garlan, D., R.T. Monroe, and D. Wile. Acme: Architectural Description of Component-Based Systems, in Foundations of Component-Based Systems, G.T. Leavens and M. Sitaraman, Editors. 2000, Cambridge University Press. pp. 47-68

[63] University of Illinois at Urbana-Champaign. The Illinois GRACE Project (Global Resource Adaptation through CoopEration). Available from: http://rsim.cs.uiuc.edu/grace/index.html [accessed: May 23, 2006].

[64] Georgia Institute of Technology. The Cyberguide project. Available from:www.cc.gatech.edu/fce/cyberguide/ [accessed: May 23, 2006].

[65] Gu T., Wang X.H, Pung H.K. and Zhang D.Q.,"An Ontology-Based Context Model in Intelligent Environments"Available from: http://citeseer.ist.psu.edu/cache/papers/cs/31620/http:zSzzSzwww.comp.nus.edu.sgzSz~gutaozSzgutao_NUSzSzCNDS2004_gutao.PDF/an-ontology-based-context.pdf[Accessed:June 27, 2007].

[66] Capra, L., Emmerich W, and Mascolo C. CARISMA: Context-Aware Reflective middleware System for Mobile Applications. IEEE Transactions on Software Engineering, 2003. 29(10): pp.929-945.

[67] Han R., Bhagwat P., LaMaire R., Mummert T. Perret V., Rubas J.,"Dynamic Adaptation in an Image Transcoding Proxy for Mobile Web Browsing". In IEEE Personal Communication, Vol. 5(6), pp. 8-17, 1998. Available from http://www.cs.colorado.edu/~rhan/IEEEPersComm_Dec98.pdf[Accessed, 16th August 2007]

[68] David, P.-C. and T. Ledoux. Towards a framework for self-adaptive component-based applications. In proceedings of Distributed applications and Interoperable Systems (DAIS'2003). 2003. Paris, France. Available from: http://pcdavid.net/research/papers/2003/dais/david-ledoux_dais2003.pdf[accessed 10th October, 2007]

[69] Zachariadis, S., C. Mascolo, and W. Emmerich. SATIN: a component model for mobile self organization.In proceedings of Proceedings of CoopIS, DOA and ODBASE. Cyprus. 2004. Available from:

[70] Doyle, R.P. and Chase J.S. Model-based resource provisioning in a web service utility. In proceedings of Fourth USENIX Symposium on Internet Technologies and Systems. 2003. Available from: http://issg.cs.duke.edu/publications/mbrp-usits03.pdf[Accessed 10th October, 2007]

[71] FIP TC-2 Workshop on Architecture Description Languages (WADL), World Computer Congress. 2004. Toulouse, France.

[72] Raverdy, P.-G. And R. Lea. DART: A distributed adaptive run-time. In proceedings of IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98). 1998.

[73] Silva F.J., Kon F., Yonder J., and Johnson R." A Pattern Language for Adaptive Distributed Systems", Technical Report, The department of Informatics, Computer Science Universities of Manhoa, Sao Paulo, and Illinois Urbana Champaign, Brazil and USA respectively. 2004.

[74] Alia M., Wold Eide V.S.,Paspallis N.,Hallsteinsen .O.,Papadopoulos G.A ," A Utility-based Adaptivity Model for Mobile Applications", 21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07) pp. 556-563, Available from: http://doi.ieeecomputersociety.org/10.1109/AINAW.2007.64, [Accessed: 03 October,2007].

[75] Indulska J., Loke S. Rakotonirainy A., and Zaslavsky A.. "Adaptive Enterprise Architecture for Mobile Computation". in Workshop on Reflective Middleware, at Middleware 2000. 2000

[76] Litiu R. and .Prakash A., "DACIA: A Mobile Component Framework for Building Adaptive Distributed Applications", ACM SIGOPS Operating Systems Review, Vol. 35, Issue 2, 2001.

[77] Jain P. and Schmidt C.S., Service Configurator: "A Pattern for Dynamic Configuration of Services" in proceedings of the third USENIX Conference on Object-Oriented Technologies(COOTS) ,June 1997. Available From: http://citeseer.ist.psu.edu/cache/papers/cs/614/http:zSzzSzsiesta.cs.wustl.eduzSz~schmidtzSzService-Configurator.pdf/jain96service.pdf[accessed:10th October, 2007].

[78] Testing Models for helping Developing Country Entrepreneurs at ground level: Lessons learned from the ECHO pilot in South Africa

Available from: http://www.bridges.org[accessed last 25th May, 2007]

[79] Otebolaku A.M., Adigun M.O., and Emuoyibofarhe J.O. "A Dynamic and Asynchronous Interface Pattern" Southern Africa Telecommunications Networks And Application Conference(SATNAC, 2006), Cape Town.

[80] Volker ,G.M. and Bershad B.N. Mobisaic : An Information System for a Mobile Wireless Computing Environment and Engineering. In proceedings of the IEEE Workshop on Mobile and Applications, 1994, Santa Cruz.

[81] Otebolaku, A.M., Iyilade J.S., Adigun M.O., "CAAM: A Context Aware Adaptation Model for Mobile Grid Service," The 11th IEEE International Conference on Computational Science and Engineering (CSE'08), 2008 pp 419-425.

APPENDIX A

Screenshot of the context monitoring log

APPENDIX B

Some Sample Code

The codes listed here contain the implementation of major components of the work. Some of the other codes are left out because of space.

(a) **Adaptation Package**: This is the package that is responsible for execution of Adaptation using reconfiguration process.

```
package AdaptationManager.adaptation;
import org.istmadam.configuration.IApplicationReconfiguration;
import org.istmadam.context.IContextAccess;
import org.istmadam.core.IComponentManagement;
import org.istmadam.connectable.IConnectable;


public class AdaptationManager implements IConnectable
{

   static public final String COMPONENT_MANAGEMENT_PORT = "ComponentManagement_Port";
   static public final String CONFIGURATION_PORT = "Configuration_Port";
   static public final String CONTEXT_ACCESS_PORT = "ContextAccess_Port";

   AdaptationCoordinator coordinator;

   public AdaptationManager()
   {
      coordinator = new AdaptationCoordinator();
   }

   public AdaptationCoordinator getAdaptationCoordinator()
   {
      return coordinator;
   }

      public void addConnection(String portName, IConnectable object)
   {
      if (portName.equals(COMPONENT_MANAGEMENT_PORT))
      {
         coordinator.setComponentManager((IComponentManagement) object);
      }
      else if (portName.equals(CONFIGURATION_PORT))
      {
         coordinator.setConfigurator((IApplicationReconfiguration) object);
      }
      else if (portName.equals(CONTEXT_ACCESS_PORT))
      {
         coordinator.setContextAccess((IContextAccess) object);
```

```
    }
  }

  public void removeConnection(String portName, IConnectable object)
  {
    throw new UnsupportedOperationException("Not implemented");
  }


}
```

(b) The reconfiguration Package

```
package Reconfigurator;

import java.util.HashMap;

import org.istmadam.core.*;


public class Reconfigurator
{
  /* The template use in the current running application, or null if no application is running yet
   */
  ConfigurationTemplate template = null;

 CACIPName componentInstance = null;
  HashMap configMap = new HashMap();
  Configuration coordinatorConfig = null;


  publicReconfiguration()
  {
  }

  public void addConfigurationForRole(String roleName, Configuration config)
  {
    configMap.put(roleName, config);
  }

  public void removeConfigurationForRole(String roleName)
  {
    configMap.remove(roleName);
  }

  public cacipName getInstanceForRole(String roleName)
  {
    //return (CACIPName)roleToInstanceMap.get(roleName);
    return getConfigurationForRole(roleName).getComponentInstance();
  }

  public Configuration getConfigurationForRole(String roleName)
  {
    return (Configuration) configMap.get(roleName);
  }
```

```
    /**
     * @return Returns the componentInstance.
     */
    public CACIPName getComponentInstance()
    {
        return componentInstance;
    }

    public void setComponentInstance(CACIPName compInstance, boolean clearRoleConfigurations)
    {
        this.componentInstance = compInstance;
        if (clearRoleConfigurations)
        {
            configMap.clear();
        }
    }

    /**
     * Get the current template used for this configuration.
    public ConfigurationTemplate getTemplate()
    {
        return template;
    }

    /**
     * @param template The template to set.
     */
    public void setTemplate(ConfigurationTemplate template)
    {
        this.template = template;
    }

    public Configuration getCoordinatorConfiguration()
    {
        return coordinatorConfig;
    }

    public void setCoordinatorConfiguration(Configuration config)
    {

    }
}
```

(c) **The eventManager Package**: is responsible for communication among the model components

```
package context.eventManager;

import org.istmadam.context.ContextElement;

import java.util.EventObject;
import java.io.Serializable;

public class ValueChangedEvent extends EventObject implements Serializable
{
```

```
public static final int CONTEXT_ELEMENT_CHANGE_UNKNOWN = 0x00;
public static final int CONTEXT_ELEMENT_ADDED        = 0x01;
public static final int CONTEXT_ELEMENT_UPDATED      = 0x02;
public static final int CONTEXT_ELEMENT_REMOVED      = 0x03;

private final ContextElement contextElement;//todo transient?
private final int changeType;

 public ValueChangedEvent(final Object source, final ContextElement contextElement)
{
   this(source, contextElement, CONTEXT_ELEMENT_CHANGE_UNKNOWN);
}

public ValueChangedEvent(final Object source, final ContextElement contextElement, final int type)
{
   super(source);
   this.contextElement = contextElement;
   this.changeType = type;
}

public ContextElement getContextElement()
{
   return contextElement;
}

public int getChangeType()
{
   return changeType;
}

public String getChangeTypeAsString()
{
   switch(changeType)
   {
      case CONTEXT_ELEMENT_ADDED:
         return "CONTEXT_ELEMENT_ADDED";
      case CONTEXT_ELEMENT_UPDATED:
         return "CONTEXT_ELEMENT_UPDATED";
      case CONTEXT_ELEMENT_REMOVED:
         return "CONTEXT_ELEMENT_REMOVED";
      case CONTEXT_ELEMENT_CHANGE_UNKNOWN:
      default:
         return "CONTEXT_ELEMENT_CHANGE_UNKNOWN";
   }
}

public String toString()
{
   StringBuffer stringBuffer = new StringBuffer("ValueChangedEvent {type:
").append(getChangeTypeAsString());

   stringBuffer.append(", element: ").append(contextElement.toString()).append("}");

   return stringBuffer.toString();
}
```

```
}
```

(c) Context Monitor :this package is responsible for context Monitoring, it contains the context sensors

```
package contextMonitor;

import context.event.ValueChangedEvent;
import org.istmadam.context.IContextListener;

public class DefaultContextSensor implements ContextSensor
{

   protected IContextListener listener;

   private final String sensorId;

   protected final Metadata metadata;

   public DefaultContextSensor(
         final String id,
         final String contextElementID,
         final IContextListener listener,
         final Metadata metadata)
   {
      if(id == null || contextElementID == null)
      {
         throw new NullPointerException("Invalid null argument");
      }

      this.sensorId = id;
      this.contextElementId = contextElementID;
      this.listener = listener;
      this.metadata = metadata;
   }

   public ContextSensor(
         final String id,
         final String contextElementID,
         final IContextListener listener)
   {
      this(id, contextElementID, listener, new DefaultMetadata());
   }

   public DefaultContextSensor(final String id, final String contextElementId)
   {
      this(id, contextElementId, null);
   }

   public synchronized void setContextChangeListener(IContextListener listener)
   {
      this.listener = listener;
   }


   public synchronized void unsetContextChangeListener(IContextListener listener)
   {
```

```
        if (this.listener == listener)
        {
            this.listener = null;
        }
    }

    public String getSensorID()
    {
        return sensorId;
    }

    public String getContextElementID()
    {
        return contextElementId;
    }

    public void fireContextChangeEvent(ValueChangedEvent valueChangedEvent)
    {
        if(listener != null)
        {
            listener.contextValueChanged(valueChangedEvent);
        }
    }

    public Metadata getMetadata()
    {
        return metadata;
    }
}
```

(d) The Evaluator Package

```
package Evaluator;

import java.util.Collection;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;
import Reconfiguration.ConfigurationTemplate;
import Reconfiguration.IConfiguration;
import context.IContextAccess;
import context.IContextListener;
import context.ContextElement;
import context.event.ValueChangedEvent;
import org.istmadam.core.IComponentManagement;
import org.istmadam.plan.BlueprintPlan;
import org.istmadam.plan.CompositionPlan;
import org.istmadam.plan.IPlan;
import org.istmadam.plan.Role;
import org.istmadam.property.PropertyEvaluator;
import org.istmadam.application.ApplicationManager;
public class AdaptationCoordinator implements IAdaptationManagement, IContextListener
{
    private IConfiguration configurator;
    //private BuilderAndPlanner planner;
```

```
    HashMap planners = new HashMap();
    HashMap oldTemplates = new HashMap();
    HashMap currentServices = new HashMap();
//    private boolean needsRebuild = true;
    private IContextAccess contextAccess;

    //public AdaptationCoordinator(BuilderAndPlanner planner)
    public AdaptationCoordinator()
    {
      //this.planner = planner;
    }

    public void setConfigurator(IConfiguration config)
    {
      configurator = config;
    }

    public void setContextAccess(IContextAccess access)
    {
      contextAccess = access;
    }

    private IComponentManagement core;
    public void setComponentManager(IComponentManagement mngr)
    {

      core = mngr;
    }

    public CACIPName launch(CACIPName mainComponentType)
    {
      // Create a planner for the new component to launch
      BuilderAndPlanner p = new BuilderAndPlanner();
      p.setComponentManager(core);
      p.buildFrameworkModel(mainComponentType);
      planners.put(mainComponentType, p);

      adjustContextListeners();
//      adjustContextListeners(p.getContextDependencies());

      CACIPName serv = evaluateAndSelect(mainComponentType);
      updateAppStatus();
      return serv;
      //return currentService;
    }

    private void updateAppStatus() {

      Set keys = currentServices.keySet();
      String[] typeStrings = new String[keys.size()];
      int strIndex = 0;
      for (Iterator nameIt = keys.iterator(); nameIt.hasNext();) {
        typeStrings[strIndex] = ((CACIPName)nameIt.next()).toString();
          strIndex++;
          }
```

```
        ApplicationManager.setRunningApplications(typeStrings);
    }

    private Set crContextListeners = new HashSet();

//  private void adjustContextListeners(Set elementNames)
    private void adjustContextListeners()
    {
        Collection ps = planners.values();
        Set elementNames = new HashSet();
        for (Iterator plannerIt = ps.iterator(); plannerIt.hasNext();) {
            BuilderAndPlanner p = (BuilderAndPlanner)plannerIt.next();
            elementNames.addAll(p.getContextDependencies());
        }

        for (Iterator oldCtxt = crContextListeners.iterator(); oldCtxt.hasNext();)
        {
            // First, stop listening to elements on which we no longer depend
            String old = (String) oldCtxt.next();
            if (!elementNames.contains(old))
            {
                System.out.println("Context listener removed: " + old);
                contextAccess.removeContextListener(this, old);
            }
        }

        for (Iterator newCtxt = elementNames.iterator(); newCtxt.hasNext();)
        {
            // Next, start listening to elements we do not already listen to
            String newEl = (String) newCtxt.next();
            if (!crContextListeners.contains(newEl))
            {
                System.out.println("Context listener added: " + newEl);
                contextAccess.addContextListener(this, newEl);
            }
        }
        crContextListeners = elementNames;
    }

//  CACIPName currentService = null;

    private void evaluateAndSelectForAll()
    {
        for (Iterator names = planners.keySet().iterator(); names.hasNext();)
        {
            evaluateAndSelect((CACIPName)names.next());
        }
    }

    private CACIPName evaluateAndSelect(CACIPName mainComponentType)
    {
        // First find planner and old template for component
        BuilderAndPlanner planner = (BuilderAndPlanner)planners.get(mainComponentType);
        ConfigurationTemplate oldTemplate = (ConfigurationTemplate)oldTemplates.get(mainComponentType);
```

```
      IEnumeratorWithReset templates = planner.buildTemplates();
      ConfigurationTemplate bestTemplate = null;
      double bestUtil = 0.0;

      ContextElement context = contextAccess.getRootContext();

      // First, reevaluate the utility of the current service, if any
      double oldUtil = 0.0;
      if (oldTemplate != null)
        oldUtil = ((Double) oldTemplate.evaluate(PropertyEvaluator.UTILITY_PROPERTY,
context)).doubleValue();

      // Iterate through the available templates, and find the one with the best utility
      while (templates.hasMoreElements())
      {
        ConfigurationTemplate crTemplate = (ConfigurationTemplate) templates.nextElement();

        double crUtil = ((Double) crTemplate.evaluate(PropertyEvaluator.UTILITY_PROPERTY,
context)).doubleValue();
        if (crUtil > bestUtil)
        {
          bestUtil = crUtil;
          bestTemplate = crTemplate;
        }
      }

      if ((bestTemplate != null) && (bestUtil > oldUtil))
      {
        System.out.println("----------NEW CONFIGURATION SELECTED - RECONFIGURATION
STARTING----------");
        // Display the plan
        System.out.println("New configuration: ");
        System.out.print("  ");
        displayTemplate(bestTemplate, "  ");
        // Change to new template
        currentServices.put(mainComponentType, configurator.configure(bestTemplate));
        oldUtil = bestUtil;
        //oldTemplate = bestTemplate;
        oldTemplates.put(mainComponentType, bestTemplate);
        System.out.println("----RECONFIGURATION COMPLETE FOR: " +
mainComponentType.toString() + " ----");
      }
      else
      {
        System.out.println("----OLD CONFIGURATION STILL BEST - NO CHANGE FOR: " +
mainComponentType.toString() + " ----");
      }
      return (MadamName)currentServices.get(mainComponentType);
    }

    public void displayTemplate(ConfigurationTemplate template, String indent)
    {
      IPlan plan = template.getPlan();
      if (plan instanceof CompositionPlan)
      {
```

```
        System.out.println("Component type: " + plan.getComponentType());
        CompositionPlan cPlan = (CompositionPlan) plan;
        Role[] roles = cPlan.getRoles();
        System.out.println(indent + "Roles: ");
        for (int i = 0; i < roles.length; i++)
        {
          String roleName = roles[i].getName();
          System.out.print(indent + "  " + roleName);
          displayTemplate(template.getChildTemplateForRole(roleName), indent + "  ");
        }
      }
      else if (plan instanceof BlueprintPlan)
      {
        BlueprintPlan bPlan = (BlueprintPlan) plan;
        System.out.println("Component type: " + plan.getComponentType() + " bp: " +
bPlan.getBlueprintName());
      }
      else
      {
        System.out.println("Component type: " + plan.getComponentType());
      }
    }

    public void contextValueChanged(ValueChangedEvent vEvent)
    {
      evaluateAndSelectForAll();
    }
}

Package BuilderAndPlanner
import java.util.Collection;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;

import org.istmadam.configuration.ConfigurationTemplate;
import org.istmadam.core.IComponentManagement;
import org.istmadam.plan.CompositionPlan;
import org.istmadam.plan.IPlan;
import org.istmadam.plan.Role;
import org.istmadam.property.PropertyEvaluator;

public class BuilderAndPlanner implements IVariantManagement
{
    private CACIPName rootType;
    private HashMap frameworkModel = new HashMap();
    private HashSet contextDependencies = new HashSet();

    private IComponentManagement core;

    public void setComponentManager(IComponentManagement mngr)
    {
      core = mngr;
    }
```

```
public void buildFrameworkModel(CACIPName component)
{
  rootType = component;

  // First, clear old model
  frameworkModel.clear();
  contextDependencies.clear();

  // Now, perform recursive buildup of model
  fillModel(component);
}

public Set getContextDependencies()
{
  return contextDependencies;

}

private HashMap emptyPropertyMap = new HashMap();

private void fillContextDependencies(IPlan plan)
{
  try
  {
    // Find context dependencies
    Collection vals = plan.getProperties().values();
    for (Iterator it = vals.iterator(); it.hasNext();)
    {
      PropertyEvaluator eval = (PropertyEvaluator) it.next();
      String[] deps = eval.getContextDependencies();
      for (int i = 0; i < deps.length; i++)
      {
        contextDependencies.add(deps[i]);
      }
    }
  }
  catch (Exception ex)
  {
    System.out.println("Exeception during BuilderAndPlanner.fillContextDependencies()");
  }
}

private void fillModel(CACIPName compType)
{
  try
  {

    Set allPlans = core.findComponentPlans(compType, emptyPropertyMap);
    frameworkModel.put(compType, allPlans);
    for (Iterator i = allPlans.iterator(); i.hasNext();)
    {
      IPlan crPlan = (IPlan) i.next();

      fillContextDependencies(crPlan);
```

```
            if (crPlan instanceof CompositionPlan)
            {

              CompositionPlan cPlan = (CompositionPlan) crPlan;
              Role[] allRoles = cPlan.getRoles();
              for (int r = 0; r < allRoles.length; r++)
              {
                CACIPName roleTypeName = allRoles[r].getComponentType();
                // Continue recursivly if component type not already handled
                if (!frameworkModel.containsKey(roleTypeName))
                {
                  fillModel(roleTypeName);
                }
              }
              CACIPName coord = cPlan.getCoordinatorType();
              if (!frameworkModel.containsKey(coord))
              {
                fillModel(coord);
              }
            }
          }
        }
        catch (CACIPException e)
        {
          e.printStackTrace();
        }
    }

    public IEnumeratorWithReset buildTemplates()
    {
      return new PlansForTypeEnumerator(rootType);
    }


    private boolean includes(Object[] objMat, Object obj)
    {
      for (int i = 0; i < objMat.length; i++)
      {
        if ((objMat[i] != null) && (objMat[i].equals(obj)))
        {
          return true;
        }
      }
      return false;
    }



    class PlansForTypeEnumerator implements IEnumeratorWithReset
    {
      IEnumeratorWithReset planVarEnums[];
      int crIndex = 0;

      public PlansForTypeEnumerator(MadamName compType)
```

```
{

   Set allPlans = (Set) frameworkModel.get(compType);
   planVarEnums = new IEnumeratorWithReset[allPlans.size()];
   int j = 0;
   for (Iterator i = allPlans.iterator(); i.hasNext();)
   {
      IPlan crPlan = (IPlan) i.next();
      if (crPlan instanceof CompositionPlan)
      {
         planVarEnums[j] = new CompositionPlanVariantEnumerator((CompositionPlan) crPlan);

      }
      else
      {
         planVarEnums[j] = new SimplePlanVariantEnumerator(crPlan);
      }
      j++;
   }
}

public void reset()
{

   for (int i = 0; i < planVarEnums.length; i++)
   {
      planVarEnums[i].reset();
   }
   crIndex = 0;
}

public boolean hasMoreElements()
{

   for (int i = crIndex; i < planVarEnums.length; i++)
   {
      if (planVarEnums[i].hasMoreElements())
      {
         return true;
      }
   }
   return false;
}

public Object nextElement()
{

   while (crIndex < planVarEnums.length)
   {
      if (planVarEnums[crIndex].hasMoreElements())
      {
         return planVarEnums[crIndex].nextElement();
      }
      else
      {
```

```
            crIndex++;
          }
        }
        return null;
    }
}

class SimplePlanVariantEnumerator implements IEnumeratorWithReset
{
    ConfigurationTemplate template;
    boolean hasMore = true;

    public SimplePlanVariantEnumerator(IPlan plan)
    {
        template = new ConfigurationTemplate(plan);
    }


    public void reset()
    {
        hasMore = true;
    }


    public boolean hasMoreElements()
    {
        return hasMore;
    }


    public Object nextElement()
    {
        if (hasMore)
        {
            hasMore = false;
            return template;
        }
        else
        {
            return null;
        }
    }

}

class CompositionPlanVariantEnumerator implements IEnumeratorWithReset
{

    CompositionPlan plan;
    Role roles[];
    PlansForTypeEnumerator roleEnums[]; // An array containing an enumerator for each role
    ConfigurationTemplate roleTemplates[]; // The last results from using nextElement() for each role
    ConfigurationTemplate coordinatorTemplate = null;
```

```
public CompositionPlanVariantEnumerator(CompositionPlan plan)
{
  this.plan = plan;

  roles = plan.getRoles();
  roleEnums = new PlansForTypeEnumerator[roles.length];
  roleTemplates = new ConfigurationTemplate[roles.length];
  for (int i = 0; i < roles.length; i++)
  {
    roleEnums[i] = new PlansForTypeEnumerator(roles[i].getComponentType());
    roleTemplates[i] = (ConfigurationTemplate) roleEnums[i].nextElement();
  }
  if (roles.length > 0)
    roleEnums[0].reset();
  Object[] coordPlans = ((Set) frameworkModel.get(plan.getCoordinatorType())).toArray();
  if (coordPlans.length > 0)
  {
    coordinatorTemplate = new ConfigurationTemplate((IPlan) coordPlans[0]);
  }
}

public void reset()
{
  for (int i = 0; i < roleEnums.length; i++)
  {
    roleEnums[i].reset();
    roleTemplates[i] = (ConfigurationTemplate) roleEnums[i].nextElement();
  }
  if (roles.length > 0)
    roleEnums[0].reset();
}

public boolean hasMoreElements()
{

  if (coordinatorTemplate == null)
  {
    return false;
  }

  for (int i = 0; i < roleTemplates.length; i++)
  {
    if (roleTemplates[i] == null)
      return false;
  }


  for (int i = 0; i < roleEnums.length; i++)
  {
    if (roleEnums[i].hasMoreElements())
    {
      return true;
    }
  }
  return false;
```

```
    }

    public Object nextElement()
    {
      if (!hasMoreElements())
      {
        return null;
      }


      for (int i = 0; i < roleEnums.length; i++)
      {
        if (roleEnums[i].hasMoreElements())
        {
          roleTemplates[i] = (ConfigurationTemplate) roleEnums[i].nextElement();
          // All ready, so break out of the iteration
          break;
        }
        else
        {
          roleEnums[i].reset();
          roleTemplates[i] = (ConfigurationTemplate) roleEnums[i].nextElement();

        }
      }
      ;

      HashMap map = new HashMap();
      for (int i = 0; i < roleEnums.length; i++)
      {
        map.put(roles[i].getName(), roleTemplates[i]);
      }

      return new ConfigurationTemplate(plan, map, coordinatorTemplate);
    }
  }
}
```